

目录

Introduction	0
项目介绍	1
新手上路	2
初始设置	2.1
安装工具链	2.2
MAC OS	2.2.1
Linux	2.2.2
高级Linux	2.2.2.1
Windows	2.2.3
代码编译	2.3
合作开发	2.4
概念解读	3
飞行模式/操作	3.1
结构框架	3.2
飞行控制栈	3.3
中间件	3.4
混控输出	3.5
PWM限制状态机	3.6
教程	4
地面站	4.1
编写应用程序	4.2
QGC的视频流	4.3
光流和LiDAR-Lite	4.4
综合测试	4.5
户外光流	4.6
多旋翼PID调参	4.7
sdlog2	4.8
ecl EKF	4.9
Preflight Checks	4.10
仿真	5

基本仿真	5.1
Gazebo仿真	5.2
硬件在环仿真	5.3
连接到ROS	5.4
自驾仪的硬件	6
Crazyfile 2.0	6.1
Intel Aero	6.2
Pixfalcon	6.3
Pixhawk	6.4
Pixracer	6.5
树莓派Pi	6.6
晓龙	6.7
光流	6.7.1
Snapdragon Advanced	6.7.2
获取I/O数据	6.7.2.1
相机和光流	6.7.2.2
中间件及架构	7
uORB	7.1
自定义MAVLink消息	7.2
守护进程	7.3
驱动框架	7.4
机型	8
统一的基础代码	8.1
添加一个新的机型	8.2
多旋翼	8.3
电机映射	8.3.1
QAV 250 Racer	8.3.2
Matrice 100	8.3.3
QAV-R	8.3.4
直升机	8.4
Wing Wing Z-84	8.4.1
垂直起降飞行器	8.5
垂直起降测试	8.5.1
TBS Caipiroshka	8.5.2

船舶, 潜水艇, 飞艇, 车辆	8.6
Companion Computers	9
Pixhawk family companion	9.1
使用DroneKit的机器人	10
DroneKit的使用	10.1
使用ROS的机器人	11
用Linux进行外部控制	11.1
在树莓派Pi2上安装ROS	11.2
MAVROS(ROS上的MAVLink)	11.3
MAVROS外部控制例程	11.4
外部位置估计	11.5
Gazebo Octomap	11.6
传感器和执行机构总线	12
I2C	12.1
SF 1XX lidar	12.1.1
UAVCAN	12.2
UAVCAN Bootloader	12.2.1
UAVCAN 固件升级	12.2.2
UAVCAN 配置	12.2.3
PWM / GPIO	12.3
UART	12.4
uLanding Radar	12.4.1
调试以及高级主题	13
FAQ	13.1
系统控制台	13.2
系统启动	13.3
参数 & 配置	13.4
自驾仪调试	13.5
仿真调试	13.6
发送调试的值	13.7
室内 / 假 GPS	13.8
相机触发器	13.9
Logging	13.10

飞行日志分析	13.11
EKF2的Log文件回放	13.12
System-wide Replay	13.13
安装Intel RealSense R200的驱动	13.14
设置云台控制	13.15
切换状态估计器	13.16
Out of tree Modules	13.17
ULog文件模式	13.18
Licenses	13.19
软件更新	14
STM32_BootLoader	14.1
Testing and CI	15
Docker 容器	15.1
Continuous Intergration	15.2
Jenkins持续集成环境	15.2.1

PX4中文维基

敬启者：

PX4开发者中文官网，翻译自 <http://dev.px4.io/>

欢迎志同道合的伙伴共同努力。

与官网相同，与官网不同，欢迎使用评论系统以及在线编辑功能。

- **GitBook**

与官网的方式相同，我们也是将网站以GitBook的方式呈现给大家。 Gitbook是一个命令行工具，可以把你 Markdown 文件汇集成电子书，并提供 PDF 等多种格式输出。你可以把 Gitbook 生成的 HTML 发布出来，就形成了一个简单的静态网站，就像现在你所看到的。用 CSDN 博客的相信大家都不会陌生 Markdown 这个工具。不多说，[点进来](#)你就知道怎么使用了。

- **Github**

这里我已经[将 GitBook 托管到了 Github 上](#)，大家感兴趣的可以 fork 下来一起完成这项工程谈不上浩大的工作，我想你也常常为打开官网望去满眼的英文而苦恼吧，来吧，我们可以做点什么的。fork 地址在[这里](#).

- **Git**

Git 这个工具非常重要，且简单易学有意思，不妨掌握一下，技多不压身。关于 Git 的学习点我们的擎天柱 [luoshi006](#), 还有不得不说 [廖雪峰的官方网站](#) 也是极好的，无意间发现 [Git Community Book 中文版](#), 循序渐进吧。

最后还是诚邀志同道合的同志加入 PX4 中文维基的汉化组。联系方式：QQ群: 499861916

From Fantasy

参与维护

1. 网页端编辑（**强烈推荐**：非常简单，只需三步就可以完成你的贡献）

- 在浏览在线页面时进行编辑本页，这样，只要你看到不妥的地方立马可以修改，具体如下图



- 或者也可以直接打开github中网页端进入要进行编辑的文件，如下图

The screenshot shows a GitHub repository page for 'FantasyJXF / Translation'. The file 'flight_modes_&_opeartion.md' is displayed. A red circle highlights the 'Edit this file' button in the file toolbar. The page includes standard GitHub navigation like Code, Issues, Pull requests, Wiki, Pulse, Graphs, and Settings, along with a file stats section showing 34 lines (33 sloc) and 2.76 KB.

飞行模式/操作

Flight Mode 飞行模式

<http://dev.px4.io/concept-flight-modes.html> 飞行模式定义了系统在任何给定时间的状态。用户使用远程遥控器或者QGroundControl地面站来进行飞行模式切换。

- 开始编辑：进入编辑页面后，对文件进行修改，会出现如下图问题

The screenshot shows a GitHub message: 'You need to fork this repository to propose changes.' It explains that you're not able to edit the repository directly and provides a link to learn more. A red box highlights the green 'Fork this repository and propose changes' button.

- 按照提示，点击`Fork this repository and propose changes`，进入即可编辑，如下图

This screenshot shows a GitHub repository page for 'FantasyJXF / Translation'. The repository has 1 unwatched star and 0 forks. The code editor shows a file named 'introduction.md' with the following content:

```
1 # PX4开发指南  
2 **~~人人都能玩，谁说小白勿扰的？~~**  
3  
4 本指南描述了如何在PX4系统架构工作
```

The second line of the file, which contains the commit message, is highlighted with a red box.

- 提交贡献：首先提名文件更改，如下图

This screenshot shows a 'Propose file change' dialog box. It has two input fields: 'as' and 'd'. At the bottom, there are two buttons: 'Propose file change' (highlighted with a red box) and 'Cancel'.

- 检查是否与原有版本有冲突，如果有，解决冲突再提交，没有则提交，如下图

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub pull request creation interface. At the top, there are dropdown menus for 'base fork' (FantasyJXF/Translation), 'base' (master), and 'head fork' (zhuzcz/Translation). A green checkmark indicates that the branches are 'Able to merge'. Below this, a text area contains the word 'as'. There are tabs for 'Write' and 'Preview', and a rich text editor toolbar. A large text input field has the letter 'd' typed into it. Below the input field is a placeholder: 'Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.' A note says 'Styling with Markdown is supported'. In the bottom right corner, a green button is labeled 'Create pull request', which is highlighted with a red box.

- 剩下来就是版主的事了，如果没有太大的问题，版主就可以合并分支了，到这你的对本文档的贡献就完成了。
- 本地编辑（git高级用户推荐）

相对于网页端编辑，本地编辑只是编辑在本地，后期的提交分支还是得在网页端进行，不过在此之前你得fork本项目到你的仓库。

The screenshot shows a GitHub repository page for 'FantasyJXF / Translation'. The top navigation bar includes 'Pull requests', 'Issues', 'Gist', and a 'Fork' button, which is circled in red. Below the navigation bar, the repository name is displayed. The main content area shows tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The 'Pull requests' tab is currently selected.

然后进行如下操作

```
#下载你的项目到本地
git clone https://github.com/FantasyJXF/Translation.git

#进入文件夹进行编辑即可，完成后如下操作

git add .
#这里可以看到你的更改状况

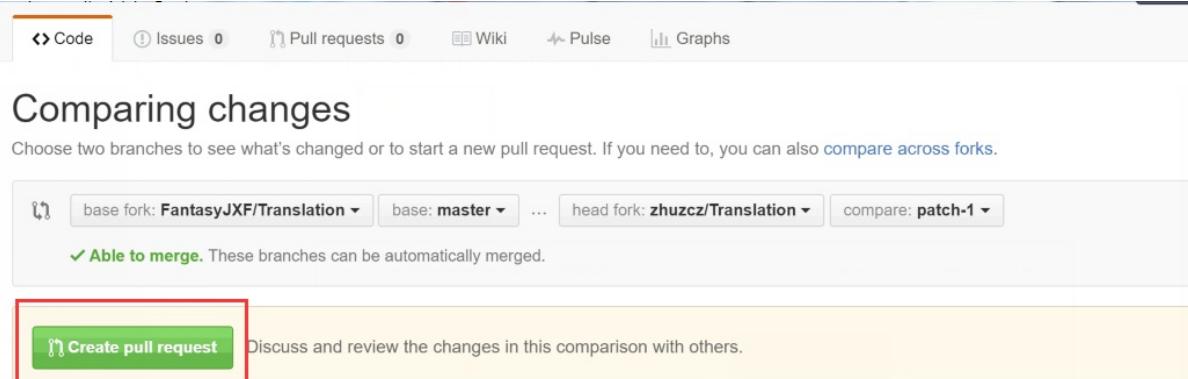
git status
#添加你的更改备注，让别人知道你干了什么

git commit -m "your comment"
#提交更改

git pull https://github.com/FantasyJXF/Translation.git master
#检查是否与FantasyJXF云端产生冲突，如果有，解决冲突后重新git commit -m "your comment"

git push origin master
#推送到个人云端
```

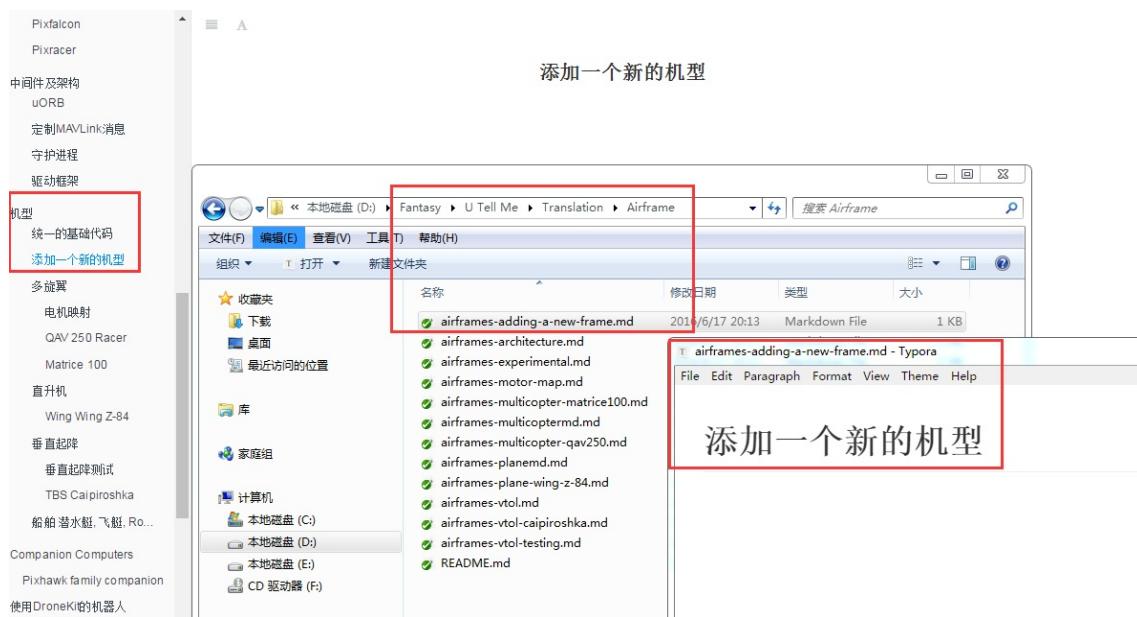
- 到目前为止，还只对你自己的仓库进行了修改，你需要 new pull request 提交分支到 FantasyJXF的仓库，如下图，可以看出，如果只是少量的更改，建议使用网页端编辑。



注意看这里

为了方便大家能够更好的参与进维护工作，现做了一些准备工作。

1. 分类。文件对应到各自的文件夹。图片集中存放。



2. 格式。当大家打开需要翻译的文件的时候，格式已经与官网一致了！关于图片、视频、.md文件的链接、引用的超链接以及代码段已经设置完毕了，大家不用再去修改。只需要将相应部分的原文进行替换即可，保证大家到手即翻，什么基础都不需要就能够轻松完成贡献。
3. 不足之处。不愿意将就，源于官网，异于官网。内容虽说大致与官网上一致，但是鉴于官网的更新难以捉摸，目前本网站的版本与官网还存在小不同，但是不影响。本网站上的视频都是Youtube上的，翻墙就能看到。关于翻墙工具，推荐Chrome浏览器 + 气星人插件

参考资料

1. 关于gitbook，可查看www.gitbook.com。
2. gitbook的官方使用，可查看<https://help.gitbook.com>。
3. 这是一个令人感动的GitBook教程<http://gitbook.zhangjikai.com/index.html>

在此感谢大家辛勤的劳动！



下表是本次汉化工作的贡献者章节分配，欢迎踊跃报名

~~标题大家有好的意见可以在线改~~

Chapter	Contributor	memo
1-项目介绍	冰	Introduction
2-新手上路	-_-	Getting Started
2.1-初始设置	冰	
2.2-安装工具链	冰	
2.2.1-MAC OS		
2.2.2-Linux	冰	
2.2.2.1-高级Linux	Innoecho	
2.2.3-Windows	风城少主	
2.3-代码编译	冰	
2.4-合作开发	冰	
3-概念解读	-_-	Concepts
3.1-飞行模式/操作	Fantasy	
3.2-结构框架	风城少主	
3.3-飞行控制栈	Fantasy	
3.4-中间件	猰�·信	

3.5-混控输出	Innoecho	
3.6-PWM限制状态机	Innoecho	
4-教程	-_-	Tutorials
4.1-地面站	范新强	
4.2-编写应用程序	PONY	
4.3-QGC的视频流	积土为山	
4.4-光流和LiDAR-Lite	PONY	
4.5-综合测试	Innoecho	
4.6-户外光流	Fantasy	
4.7-多旋翼PID调参	Fantasy	
4.8-sdlog2	Fantasy	
4.9-ecl EKF		
5-仿真	-_-	Simulation
5.1-基本仿真	积土为山	
5.2-Gazebo仿真	积土为山	
5.3-硬件在环仿真	Innoecho	
5.4-连接ROS	Innoecho	
6-自驾仪的硬件	-_-	Autopilot Hardware
6.1-Crazyfile 2.0	Fantasy	
6.2-Intel Aero		
6.3-Pixfacon	景略	
6.4-Pixhawk	景略	
6.5-Pixracer	景略	
6.6-树莓派Pi 2	誓言	
6.7-骁龙	誓言	
6.7.1-光流	Fantasy	
6.7.2-高级骁龙		
6.7.2.1-获取I/O数据		
6.7.2.1-相机和光流		
7-中间件及架构	-_-	Middleware and Architecture
7.1-uORB	彩虹小羊	
7.2-自定义MAVlink消息	彩虹小羊	

7.3-守护进程	彩虹小羊	
7.4-驱动框架	彩虹小羊	
8-机型	-_-	Airframes
8.1-统一的基础代码	Innoecho	
8.2-添加一个新的机型	Innoecho	
8.3-多旋翼	Innoecho	
8.3.1-电机映射	Innoecho	
8.3.2-QAV 250 Racer	Innoecho	
8.3.3-Matrice 100	Innoecho	
8.3.4-QAV-R	Fantasy	
8.4-直升机	Innoecho	
8.4.1-Wing Wing Z-84	Innoecho	
8.5-垂直起降飞行器	Innoecho	
8.5.1-垂直起降测试	Innoecho	
8.5.2-TBS Caipiroshka	Innoecho	
8.6-船舶，潜水艇，飞艇，racer	Innoecho	
9-Companion Computer	-_-	Companion Computers
9.1-Pixhawk family	Innoecho	
10-使用DroneKit的机器人	Innoecho	Robotics using DroneKit
10.1-DroneKit的使用	Innoecho	
11-使用ROS的机器人	-_-	Robotics using ROS
11.1-用Linux进行外部控制	Innoecho	
11.2-在树莓派Pi2上安装ROS	Innoecho	
11.3-MAVROS	Innoecho	
11.4-MAVROS外部控制例程	Innoecho	
11.5-外部位置估计	Innoecho	
11.6-Gazebo Octomap	Innoecho	
12-传感器和执行机构总线	-_-	Sensor and Actuator Buses
12.1-I2C BUS	-_-	
12.1.1-SF1XX Lidar		
12.2-UAVCAN		

12.2.1-UAVCAN Bootloader		
12.2.2-UAVCAN 固件升级		
12.2.3-UAVCAN 配置		
12.3-PWM / GPIO		
12.4-UART	-_-	
12.4.1-uLanding Radar		
13-调试以及高级主题	-_-	Debugging and Advanced Topics
13.1-FAQ	如果你永无畏惧	
13.2-系统控制台	why	
13.3-系统启动	why	
13.4-参数 & 配置		
13.5-自驾仪调试		
13.6-仿真调试		
13.7-发送调试的值		
13.8-室内 / 假 GPS		
13.9-相机触发器	含笑~饮砒霜	
13.10-Logging	老四	
13.11-飞行日志分析	Fantasy	
13.12-EKF2的Log文件回放		
13.13-System-wide Replay		
13.14-安装RealSense R200的驱动		
13.15-设置云台控制	含笑~饮砒霜	
13.16-切换状态估计器		
13.17-Out of tree Modules		
13.18-ULog文件模式		
13.19-Licenses		
14-软件更新	-_-	Software Update
14.1-STM32 BootLoader	Fantasy	
15-测试和持续集成		Testing and CI
15.1-Docker 容器		

15.2-持续集成		
15.2.1-Jenkins持续集成环境		

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

PX4开发指南

注意：仅限于开发者！这个指南旨在推动开发而不是为消费者准备。

这个指南介绍了如何在PX4系统架构下开发，它使开发人员能够：

- 了解[系统的概况](#)。
- 获取和修改[PX4飞行栈](#)和[PX4中间件](#)。
- 在[骁龙飞控](#)、[Pixhawk](#)和[Pixfalcon](#)上应用PX4。

许可证

PX4开发者文档发布为[CC BY 4.0](#)。具体请查看细节[Github Repository](#)。

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22
12:56:03

新手上路

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

初始设置

官网英文原文地址：<http://dev.px4.io/starting-initial-config.html>

在开始开发PX4之前，系统需要按照默认配置进行初始配置，以保证硬件设置合适，被检测到。下面的一个视频介绍[Pixhawk硬件](#)和[地面站](#)设置过程，[下面是支持的参考机架类型的列表](#)。

下载[DAILY BUILD版本的QGroundControl](#)并按照下面的说明来设置你的飞行器。参考[QGroundControl 教程](#)来了解任务规划，放飞和参数设置的具体细节。

下面的视频介绍一系列的设置选项

To view this video please enable JavaScript, and consider upgrading to a web browser that supports [HTML5 video](#)

无线电控制选项

PX4飞行控制栈并不强制要求无线电控制系统。也不要求使用单独的开关来选择飞行模式。

没有无线电控制的飞行

所有的无线电控制装置的检查可以通过设置参数 `COM_RC_IN_MODE` 为 `1` 禁用。这将不允许手动飞行，但是，除了比如flying in。

单通道模式开关

在这种模式下，系统将接受一个单一的通道作为模式开关，而不是使用多个开关，这在[legacy wiki](#)有解释。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

安装文件和代码

官网英文原文地址：<http://dev.px4.io/starting-installing.html>

PX4代码可以在 Mac OS, Linux 或者 Windows 上进行开发，建议在 Mac OS 和 Linux 上进行开发，因为图像处理和高级导航在 windows 上不容易开发。如果不确定，新的开发者应默认用 Linux 和当前 Ubuntu 长期支持版本（Ubuntu LTS edition）。

开发环境

开发环境安装涉及到下面几种：

- Mac OS
- Linux
- Windows

如果你对 Docker 熟悉，你可以使用其中一个容器：[Docker Containers](#)

一旦开发环境安装完成，可以接着去 [编译程序](#)。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

安装文件和代码

官网英文原文地址：<http://dev.px4.io/starting-installing-mac.html>

第一步就是从Mac应用商店中安装Xcode。安装完成后打开一个新的终端并安装命令行工具：

```
xcode-select --install
```

Homebrew Installation

推荐使用Mac OS X的[Homebrew 包管理器](#)进行安装。Homebrew的安装十分便捷：[安装指南](#)。

安装好Homebrew以后，拷贝以下命令到终端命令行：

```
brew tap PX4/homebrew-px4
brew tap osrf/simulation
brew update
brew install git bash-completion genromfs kconfig-frontends gcc-arm-none-eabi
brew install astyle cmake ninja
# simulation tools
brew install ant graphviz sdformat3 eigen protobuf
brew install homebrew/science/opencv
```

然后安装我们需要的python包：

```
sudo easy_install pip
sudo pip install pyserial empy
```

Java for jMAVSim

如果你打算使用jMAVSim，需要同时安装[Java JDK 8](#)。

骁龙飞行平台

高通为Ubuntu提供了可靠的工具链。因此使用骁龙飞行平台的开发者应该安装一个Ubuntu虚拟机，并参考Linux下方法进行工具链的安装。PX4开发团队使用的虚拟机软件是VMWare，尤其是在VMWare对USB有了稳定的 support 以后。

仿真

OS X 预装了CLANG. 因此无需再安装其他的编译器.

编辑器 / IDE

最后下载并安装 Qt Creator: [下载](#)

接下来进行 [首次编译!](#)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Linux开发环境

官网英文原文地址：<http://dev.px4.io/starting-installing-linux.html>

我们使用Debian / Ubuntu LTS 作为Linux的标准支持版本，但是也支持Cent OS 和 Arch Linux的发行版本

权限设置

警告：永远不要使用 `sudo` 来修复权限问题，否则会带来更多的权限问题，需要重装系统来解决。

把用户添加到用户组 "dialout":

```
sudo usermod -a -G dialout $USER
```

然后注销后，重新登录，因为重新登录后所做的改变才会有效。

安装

更新包列表，安装下面编译PX4的依赖包。PX4主要支持的系列：

- NuttX based hardware: [Pixhawk](#), [Pixfalcon](#), [Pixracer](#), [Crazyflie](#), [Intel Aero](#)
- Snapdragon Flight hardware: [Snapdragon](#)
- Linux-based hardware: [Raspberry Pi 2/3](#)), Parrot Bebop
- Host simulation: [jMAVSIM SITL](#) and [Gazebo SITL](#)

提示：安装[Ninja Build System](#)可以比make更快进行编译。如果安装了它就会自动选择使用它进行编译。

```
sudo add-apt-repository ppa:george-edison55/cmake-3.x -y
sudo apt-get update
sudo apt-get install python-argparse git-core wget zip \
    python-empy qtcreator cmake build-essential genromfs -y
# simulation tools
sudo apt-get install ant protobuf-compiler libeigen3-dev libopencv-dev openjdk-8-jdk open
```

基于NuttX的硬件

Ubuntu配备了一系列代理管理，这会严重干扰任何机器人相关的串口（或usb串口），卸载掉它也不会有什么影响：

```
sudo apt-get remove modemmanager
```

更新包列表和安装下面的依赖包。务必安装指定的版本的包

```
sudo apt-get install python-serial openocd \
    flex bison libncurses5-dev autoconf texinfo build-essential \
    libftdi-dev libtool zlib1g-dev \
    python-empy -y
```

在添加arm-none-eabi工具链之前，请确保删除残余。

```
sudo apt-get remove gcc-arm-none-eabi gdb-arm-none-eabi binutils-arm-none-eabi gcc-arm-em
sudo add-apt-repository --remove ppa:team-gcc-arm-embedded/ppa
```

如果 `gcc-arm-none-eabi` 版本导致PX4/Firmware编译错误，请参考[the bare metal installation instructions](#) 手动安装4.9或者5.4版本的arm-none-eabi工具链。

晓龙

工具链安装

```
sudo apt-get install android-tools-adb android-tools-fastboot fakechroot fakeroot unzip x
```

```
git clone https://github.com/ATLFlight/cross_toolchain.git
```

Get the Hexagon SDK 3.0 from QDN:

<https://developer.qualcomm.com/download/hexagon/hexagon-sdk-v3-linux.bin>

This will require a QDN login. You will have to register if you do not already have an account.

Now move the following files in the download folder of the cross toolchain as follows:

```
mv ~/Downloads/hexagon-sdk-v3-linux.bin cross_toolchain/downloads
```

Install the toolchain and SDK like this:

```
cd cross_toolchain
./installv3.sh
cd ..
```

Follow the instructions to set up the development environment. If you accept all the install defaults you can at any time re-run the following to get the env setup. It will only install missing components.

After this the tools and SDK will have been installed to "\$HOME/Qualcomm/...". Append the following to your ~/.bashrc:

```
export HEXAGON_SDK_ROOT="${HOME}/Qualcomm/Hexagon_SDK/3.0"
export HEXAGON_TOOLS_ROOT="${HOME}/Qualcomm/HEXAGON_Tools/7.2.12/Tools"
export PATH="${HEXAGON_SDK_ROOT}/gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf_linux/
```

Load the new configuration:

```
source ~/.bashrc
```

Sysroot Installation

A sysroot is required to provide the libraries and header files needed to cross compile applications for the Snapdragon Flight applications processor.

The qrlSDK sysroot provies the required header files and libraries for the camera, GPU, etc.

Download the file [Flight_3.1.1_qrlSDK.zip](#) and save it in `cross_toolchain/download/`.

```
cd cross_toolchain
unset HEXAGON_ARM_SYSROOT
./qrlinux_sysroot.sh
```

Append the following to your ~/.bashrc:

```
export HEXAGON_ARM_SYSROOT=${HOME}/Qualcomm/qrlinux_v3.1.1_sysroot
```

Load the new configuration:

```
source ~/.bashrc
```

For more sysroot options see [Sysroot Installation](#)

升级ADSP固件

在构建，烧写以及运行代码之前，还需要升级ADSP固件。

参考

[GettingStarted](#)是另外一个工具链安装向导。[HelloWorld](#)和[DSPAL tests](#)可以用来验证工具链安装和DSP镜像。

DSP的信息可以通过mini-dm查看。

```
$HOME/Qualcomm/Hexagon_SDK/2.0/tools/mini-dm/Linux_Debug/mini-dm
```

Note: Alternatively, especially on Mac, you can also use [nano-dm](#).

树莓派

树莓派开发者应该从下面地址下载树莓派Linux工具链。安装脚本会自动安装交叉编译工具链。如果想要用原生树莓派工具链在树莓派上直接编译，参见[这里](#)。

```
git clone https://github.com/pixhawk/rpi_toolchain.git
cd rpi_toolchain
./install_cross.sh
```

在工具链安装过程中需要输入密码。

如果不把工具链安装在默认位置 /opt/rpi_toolchain，可以执行 `./install_cross.sh <PATH>` 向安装脚本传入其它地址。安装脚本会自动配置需要的环境变量。

Finally, run the following command to update the environmental variables:

```
source ~/.profile
```

Parrot Bebop

Developers working with the Parrot Bebop should install the RPi Linux Toolchain. Follow the description under [Raspberry Pi hardware](#).

Next, install ADB.

```
sh sudo apt-get install android-tools-adb -y`
```

完成

继续，进行[第一次构建](#)！

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22
12:56:03

Arch及CentOS工具链安装指南

官网英文原文地址：<http://dev.px4.io/starting-installing-linux-boutique.html>

USB设备配置

Linux用户需要明确地允许JTAG编程适配器可以访问USB总线。

对于Archlinux：用`uucp`替换`plugdev`，然后执行下面命令。

在sudo模式下执行ls命令以确保之后的命令可以成功执行：

```
sudo ls
```

暂时获得sudo权限，执行以下命令：

```
cat > $HOME/rule.tmp << _EOF
# All 3D Robotics (includes PX4) devices
SUBSYSTEM=="usb", ATTR{idVendor}=="26AC", GROUP="plugdev"
# FTDI (and Black Magic Probe) Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="0483", GROUP="plugdev"
# Olimex Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="15ba", GROUP="plugdev"
_EOF
sudo mv $HOME/rule.tmp /etc/udev/rules.d/10-px4.rules
sudo /etc/init.d/udev restart
```

用户需要被添加到plugdev组：

```
sudo usermod -a -G plugdev $USER
```

小众Linux系统工具链安装指南

CentOS

构建需要Python 2.7.5支持，因此应该使用CentOS 7（当前最新版本，较早的CentOS版本可能带有python v2.7.5，但是不推荐使用，因为它可能会破坏yum）。

需要使用EPEL仓库来安装openocd，libftdi-devel和libftdi-python。

```
wget https://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
sudo yum install epel-release-7-5.noarch.rpm
yum update
yum groupinstall "Development Tools"
yum install python-setuptools
easy_install pyserial
easy_install pexpect
yum install openocd libftdi-devel libftdi-python python-argparse flex bison-devel ncurses
```

注意：你可能还想要安装python-pip和screen。

其它32位库

安装完arm工具链之后，执行以下命令测试：

```
arm-none-eabi-gcc --version
```

如果返回下列信息

```
bash: gcc-arm-none-eabi-4_7-2014q2/bin/arm-none-eabi-gcc: /lib/ld-linux.so.2: bad ELF int
```

那么你还需要安装其它的32位库：glibc.i686，ncurses-libs.i686

```
sudo yum install glibc.i686 ncurses-libs.i686
```

安装ncurses-libs.i686将会同时安装其它大部分的依赖32位库。CentOS 7将会安装大部分PX4相关设备而不需要添加任何的udev规则。预定义的'dialout'用户组可以访问这些设备，因此可以忽略添加udev规则的资料。所需要确保的仅仅是你的账户是'dialout'组的成员。

Arch Linux

```
sudo pacman -S base-devel lib32-glibc git-core python-pyserial zip python-empy
```

安装Arch User Repository (AUR)的包管理器yaourt。

然后使用它下载，编译以及安装以下内容：

```
yaourt -S genromfs
```

权限

用户需要被添加至'uucp'组：

```
sudo usermod -a -G uucp $USER
```

执行上述操作后，需要注销账户然后再次登陆。

注销账户然后再次登陆的目的是使更改生效，当然移除设备并再次插入也是必要操作。

工具链安装

执行下面的脚本来安装GCC 4.9或者5.4版本：

GCC 4.9:

```
pushd .
cd ~
wget https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update/+download/gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2
exportline="export PATH=$HOME/gcc-arm-none-eabi-4_9-2015q3/bin:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo $exportline >> . ~/.profile
popd
```

GCC 5.4:

```
pushd .
cd ~
wget https://launchpad.net/gcc-arm-embedded/5.0/5-2016-q2-update/+download/gcc-arm-none-eabi-5_4-2016q2-20160622-linux.tar.bz2
exportline="export PATH=$HOME/gcc-arm-none-eabi-5_4-2016q2/bin:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo $exportline >> . ~/.profile
popd
```

如果使用Debian Linux，执行下列命令：

```
sudo dpkg --add-architecture i386
sudo apt-get update
```

安装32位支持库（如果已经是运行在32位，那么可能会失败，则此步骤可以跳过）：

```
sudo apt-get install libc6:i386 libgcc1:i386 libstdc++5:i386 libstdc++6:i386
sudo apt-get install gcc-4.6-base:i386
```

Ninja构建系统

Ninja比Make更快，并且PX4的CMake生成器可以支持它。不幸的是，Ubuntu目前只支持一个非常过时的版本。下载二进制文件并添加到系统路径来安装最新版本的Ninja：

```
mkdir -p $HOME/ninja
cd $HOME/ninja
wget https://github.com/martine/ninja/releases/download/v1.6.0/ninja-linux.zip
unzip ninja-linux.zip
rm ninja-linux.zip
exportline="export PATH=$HOME/ninja:\$PATH"
if grep -Fxq "$exportline" ~/.profile; then echo nothing to do ; else echo $exportline >>
. ~/.profile
```

问题解答

版本测试

输入：

```
arm-none-eabi-gcc --version
```

输出应该类似以下内容：

```
arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.7.4 20140401 (release) [ARM/e
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

如果输出是：

```
arm-none-eabi-gcc --version
arm-none-eabi-gcc: No such file or directory
```

确认按照安装步骤正确安装32位库。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Windows安装指南

官网英文原文地址：<http://dev.px4.io/starting-installing-windows.html>

虽然Windows上的工具链是可用的，但官方并不支持，我们不推荐使用Windows。Windows上固件编译的过程十分缓慢，且不支持新的板子，比如骁龙（Snapdragon Flight），它也不能运行标准机器人软件包，许多开发人员使用原型计算机视觉和导航。开始在Windows上开发之前，可以考虑安装一个双启动环境 [Ubuntu](#)。

开发环境安装

下载并在系统上安装这些：

- [Qt Creator IDE](#)
- [PX4 Toolchain Installer v14 for Windows Download](#) (32/64 bit systems, complete build system, drivers)
- [PX4 USB Drivers](#) (32/64 bit systems)
- [CMake](#)(这里推荐安装CMake-3.3.2-win32-x86)

现在继续运行：[代码编译](#)！

新消息!Windows上的Bash

现在,那些想要在本地运行Bash shell进而按照Linux编译教程进行操作的Windows用户有了新的选择.请参见[BashOnWindows](#).我们已经进行了验证,PX4可以在此环境下成功编译.目前还无法用其刷写固件,但是你可以使用Mission Planner或者QGroundControl地面站刷固件.

© PX4WIKI team all right reserved , powered by Gitbook该文件修订时间：2017-01-22
12:56:03

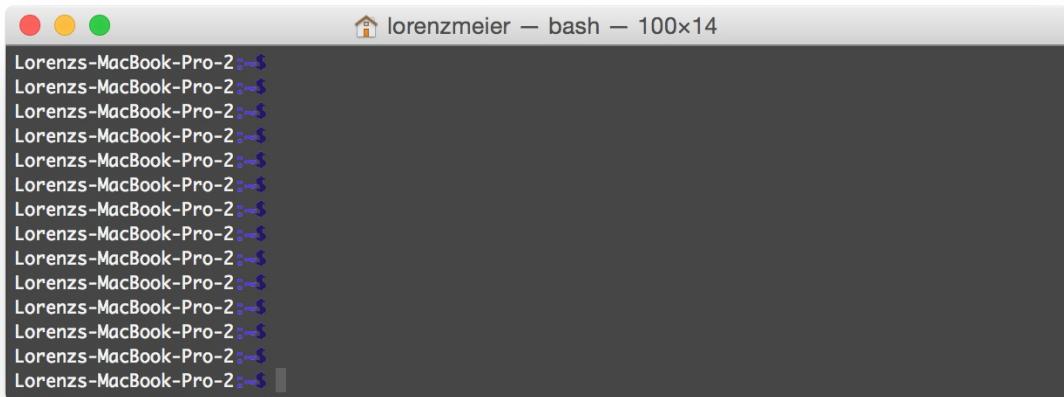
编译px4软件

官网英文原文地址：<http://dev.px4.io/starting-building.html>

PX4可以在控制台或者图形界面/IDE开发

在控制台编译

在去到图形界面或者IDE前，验证系统设置的正确性非常重要，因此打开控制台。在 OS X, 敲击 **⌘-space**，并搜索'terminal'。在Ubuntu，单击启动栏，搜索“terminal”（或者`ctrl+alt+T`）。在windows平台，在开始菜单找到px4文件夹，单击'PX4 Console'



```
Lorenzs-MacBook-Pro-2:~$  
Lorenzs-MacBook-Pro-2:~$
```

终端在Home目录启动，我们默认去到'~/src/Firmware' 然后，克隆顶层资源库。有经验的开发者可以克隆自己的复制的[资源库](#)

```
mkdir -p ~/src  
cd ~/src  
git clone https://github.com/PX4/Firmware.git  
cd Firmware  
git submodule update --init --recursive  
cd ..
```

上述命令结束以后，还不能直接对源代码进行编译，直接编译可能会出现内存溢出的错误 `error:ld returned 1 exit status`，原因是arm-none-eabi 4.7.4版本不对，直接编译失败的结果显示如下。

```
collect2.exe:error:ld returned 1 exit status
make[3]: *** [src/firmware/nuttx/firmware_muttx] Error 1
make[2]: *** [src/firmware/nuttx/CMakeFiles/firmware_muttx.dir/all] Error 2
make[1]: *** [all] Error 2
make: *** [px4fmu-v2_default] Error 2
```

这个问题可以通过下载[4.8.4版本](#)，进行解压后复制并替换掉PX4 Toolchain安装目录下Toolchain文件夹内的相应文件即可。在直接使用硬件前，推荐先[进行仿真](#)。喜欢在图形界面开发环境工作的用户也应该继续完成下面部分。

基于NuttX / Pixhawk的硬件板

```
cd Firmware
make px4fmu-v2_default
```

注意到“make”是一个字符命令编译工具，“px4fmu-v2”是硬件/ardupilot版本，“default”是默认配置，所有的PX4编译目标遵循这个规则。

成功编译的最后输出是这样的：

```
[100%] Linking CXX executable firmware_nuttx
[100%] Built target firmware_nuttx
Scanning dependencies of target build_firmware_px4fmu-v2
[100%] Generating nuttx-px4fmu-v2-default.px4
[100%] Built target build_firmware_px4fmu-v2
```

通过在命令后面添加‘upload’，编译的二进制程序就会通过USB上传到飞控硬件：

```
make px4fmu-v2_default upload
```

上传成功时输出情况如下：

```
Erase : [=====] 100.0%
Program: [=====] 100.0%
Verify : [=====] 100.0%
Rebooting.

[100%] Built target upload
```

Raspberry Pi 2 boards

The command below builds the target for Raspbian (posix_pi2_release).

```
cd Firmware
make posix_rpi2_release # for cross-compiler build
```

The "mainapp" executable file is in the directory build_posix_rpi2_release/src/firmware/posix. Copy it over to the RPi (replace YOUR_PI with the IP or hostname of your RPi, [instructions how to access your RPi](#))

```
scp build_posix_rpi2_release/src/firmware/posix/mainapp pi@YOUR_PI:/home/pi/
```

And run it with :

```
./mainapp
```

If you're building *directly* on the Pi, you will want the native build target (posix_pi2_default).

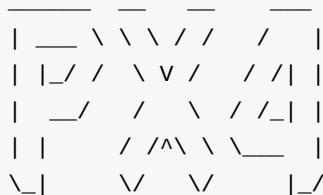
```
cd Firmware
make posix_rpi2_default # for native build
```

The "mainapp" executable file is in the directory build_posix_rpi2_default/src/firmware/posix. Run it directly with :

```
./build_posix_rpi2_default/src/firmware/posix/mainapp
```

A successful build followed by executing mainapp will give you this :

```
[init] shell id: 1996021760
[init] task name: mainapp
```



Ready to fly.

pxh>

QuRT / Snapdragon based boards

Build it

The commands below build the targets for the Linux and the DSP side. Both executables communicate via [muORB](#).

```
cd Firmware  
make eagle_default
```

To load the SW on the device, connect via USB cable and make sure the device is booted. Run this in a new terminal window:

```
adb shell
```

Go back to previous terminal and upload:

```
make eagle_default upload
```

Note that this will also copy (and overwrite) the two config files [mainapp.config](#) and [px4.config](#) to the device. Those files are stored under /usr/share/data/adsp/px4.config and /home/linaro/mainapp.config respectively if you want to edit the startup scripts directly on your vehicle.

The mixer currently needs to be copied manually:

```
adb push ROMFS/px4fmu_common/mixers/quad_x.main.mix /usr/share/data/adsp
```

Run it

Run the DSP debug monitor:

```
 ${HEXAGON_SDK_ROOT}/tools/mini-dm/Linux_Debug/mini-dm
```

Go back to ADB shell and run mainapp:

```
cd /home/linaro  
. ./mainapp mainapp.config
```

Note that the mainapp will stop as soon as you disconnect the USB cable (or if you ssh session is disconnected). To fly, you should make the mainapp auto-start after boot.

Auto-start mainapp

To run the mainapp as soon as the Snapdragon has booted, you can add the startup to

```
rc.local :
```

Either edit the file `/etc/rc.local` directly on the Snapdragon:

```
adb shell  
vim /etc/rc.local
```

Or copy the file to your computer, edit it locally, and copy it back:

```
adb pull /etc/rc.local  
gedit rc.local  
adb push rc.local /etc/rc.local
```

For the auto-start, add the following line before `exit 0`:

```
(cd /home/linaro && ./mainapp mainapp.config > mainapp.log)  
  
exit 0
```

Make sure that the `rc.local` is executable:

```
adb shell  
chmod +x /etc/rc.local
```

Then reboot the Snapdragon:

```
adb reboot
```

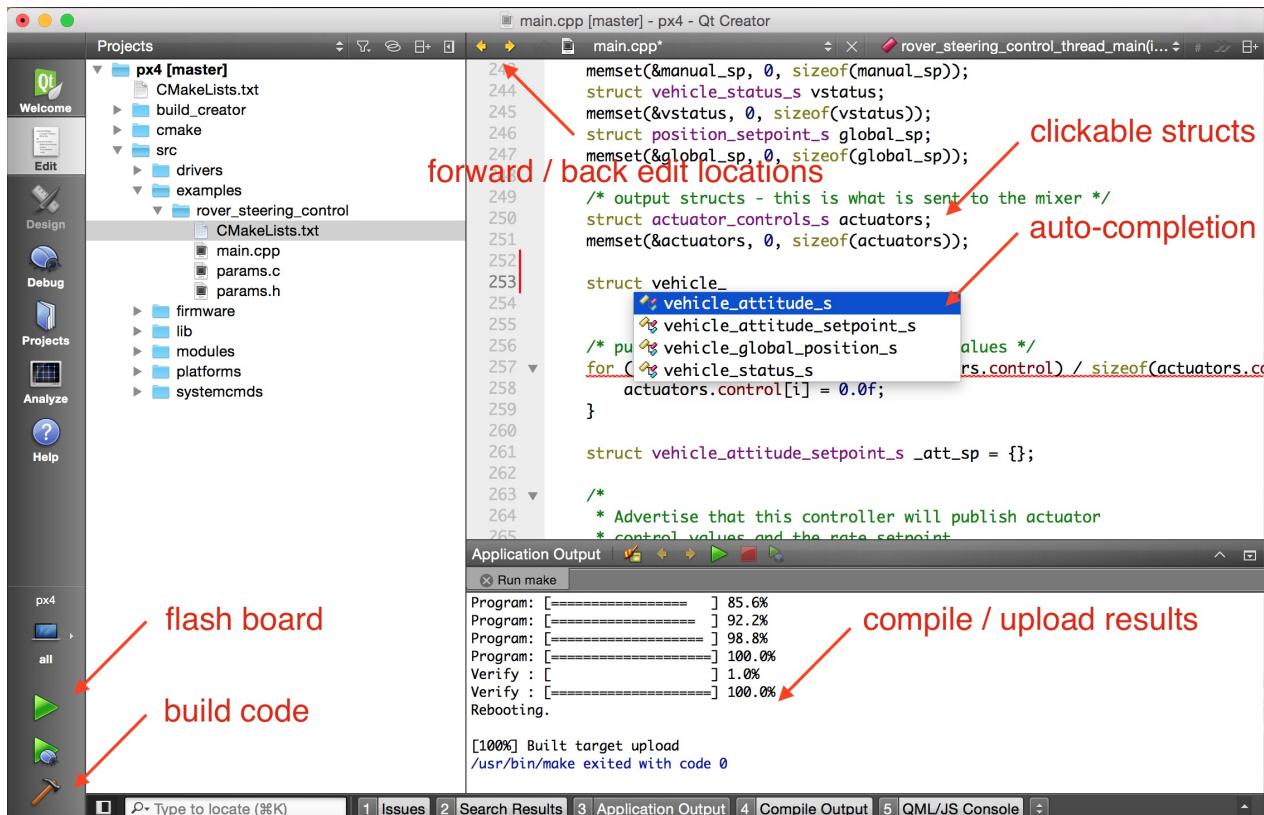
图形IDE界面下编译

PX4 支持Qt Creator, Eclipse 和Sublime Text三种集成式开发环境。 Qt Creator是最友好的开发环境，所以被是唯一官方支持的IDE。除非资深的Eclipse 或Sublime开发者，否则一般不推荐使用Eclipse或Sublime进行二次开发。硬件底层开发可以在 [Eclipse project](#) 和 [a Sublime project](#) 找到源码。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports [HTML5 video](#)

Qt Creator 功能

Qt creator 提供单击选择变量、代码自动补全、代码编译和固件上传等功能。



Linux 平台的 Qt Creator

在启动Qt creator之前，需要先创建[工程文件](#)：

```
cd ~/src/Firmware
mkdir ..../Firmware-build
cd ..../Firmware-build
cmake ..../Firmware -G "CodeBlocks - Unix Makefiles" -DCONFIG=nuttx_px4fmu-v2_default
```

接着启动Qt creator（如果系统没安装Qt Creator 百度一下linux下安装Qt Creator，然后再启动Qt Creator）并加载 Firmware 根目录下 CMakeLists.txt 文件，步骤：点击工具栏 File -> Open File or Project -> Select the CMakeLists.txt file。如果加载提示ninja没有安装，请按照“高级Linux”章节进行ninja编译工具的安装，安装完成后，log out（登出）并log in（登入）。

加载了文件后，点击左侧projects按钮，在run configuration栏选择'custom executable'，在executable 栏里输入'make'， argument栏输入 'upload'，将'play'按钮配置成运行工程。

Windows 平台的 Qt Creator

Windows平台下的Qt Creator开发目前也没经过详细测试。

Mac OS 平台的 Qt Creator

启动 Qt Creator 之前，需要先创建 [project file](#)：

```
cd ~/src/Firmware  
mkdir build_creator  
cd build_creator  
cmake .. -G "CodeBlocks - Unix Makefiles"
```

完成上述步骤以后，启动 Qt Creator，完成下面视频中的步骤，就可以进行工程文件的编译了。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports [HTML5 video](#)

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

贡献代码

官网英文原文地址：<http://dev.px4.io/starting-contributing.html>

核心开发团队和社区的联系信息可以在下面找到。PX4项目使用了三个分支Git branching model:

- **master** 默认情况下不稳定，可以看到快速的开发。
- **beta** 已充分测试，面向飞行测试者。
- **stable** 指向最新的发布分支。

我们尝试通过**rebases**保持一个线性的历史，避免**Github flow**。但是由于全球的开发队伍和快速的开发转移，我们会定期分类合并。

为了贡献新的功能，首先[注册Github账户](#)，然后[fork](#)仓库，[创建新分支](#)，加入你的改变，最后发送[\[pull request\]](#)。当它们通过我们的持续的[综合测试](#)，更新就会被合并。

所有的贡献必须在 [BSD 3-clause license](#) 许可下进行，并且所有的代码在使用上不能提出任何的，进一步的限制。

测试飞行结果

飞行测试对于保证质量非常重要，请从microSD卡上传飞行日志到 [Log Muncher](#)，并在[论坛](#)分享连接，附带有书面飞行报告。Test flights are important for quality assurance. Please upload the logs from the microSD card to [Log Muncher](#) and share the link on the [forum](#) along with a verbal flight report.

论坛和聊天

- [Google+](#)
- [Gitter](#)
- [PX4 Users Forum](#)

每周开发电话

PX4团队在每周同时进行电话联系（通过[Mumble client](#)客户端连接）。

- TIME: 19:00h Zurich time, 1 p.m. Eastern Time, 10 a.m. Pacific Standard Time
- Server: mumble.dronecode.org

- Port: 10028
- Channel: PX4 Channel
- The agenda is announced the same day on the [px4users forum](#)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

概念解读

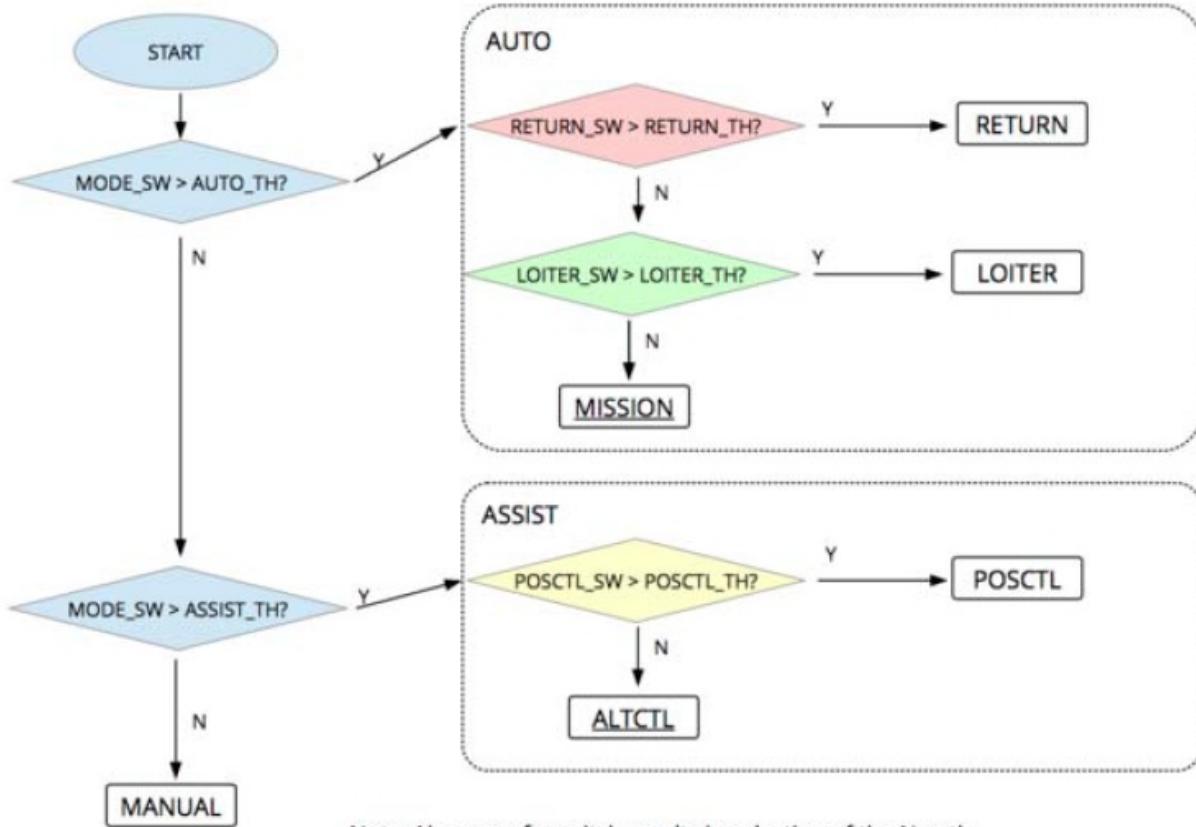
© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

飞行模式

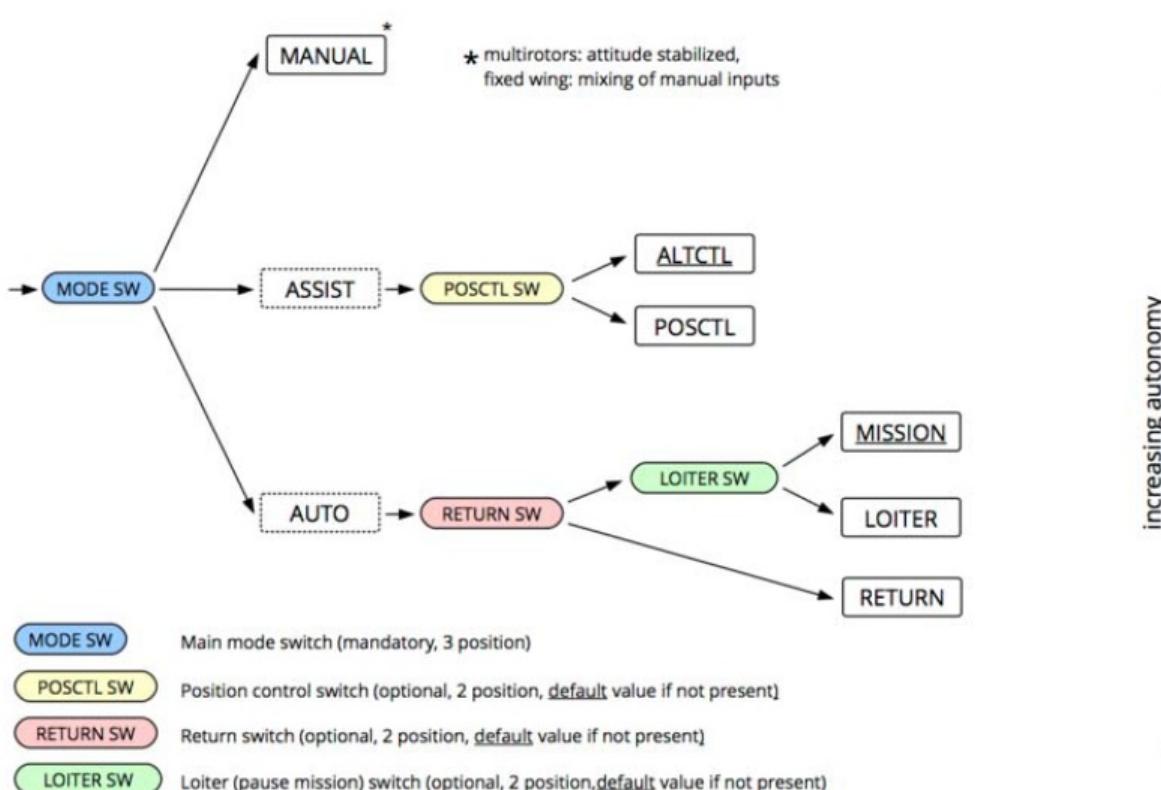
官网英文原文地址：<http://dev.px4.io/concept-flight-modes.html>

飞行模式定义了系统在任何给定时间的状态。用户可以使用远程遥控器或者**QGroundControl**地面站来进行飞行模式切换。

飞行模式的简要说明



PIXHAWK 支持6 种飞行模式具体看图



• 手动模式 MANUAL

- 固定翼飞行器\车辆\船舶：飞手的控制输入直接输出到混控器。
- 多旋翼飞行器：

- 特技模式 **ACRO**：飞手的输入被转换成横滚、俯仰、偏航角速度 传送给自驾仪。在这个模式下多旋翼飞行器可以完全翻转。油门直接输出到混控器。
- 角速度控制 **RATTITUDE**如果飞手的输入超过了设定的阈值，则将其转换成横滚、俯仰、偏航角速度命令传送给自驾仪；如果输入没有超过阈值，则将其转换成横滚、俯仰转角度以及偏航角速度命令。油门直接输出到混控器。
- 角度控制 **ANGLE**飞手的输入被转换成横滚、俯仰角度 命令以及偏航角速度 命令传送给自驾仪。油门直接输出到混控器。

• 辅助模式 **ASSISTED**

◦ 高度控制 **ALTCTL**

- 固定翼飞行器：当横滚、俯仰、偏航输入（**RPY**）都居中时（不超过一些特定的死区范围），飞行器将回到直线水平的飞行状态然后保持当前的高度。飞行器会随着风漂移。
- 多旋翼飞行器 横滚、俯仰、偏航输入与主模式中的相同。油门输入表明以预定的最大速率上升或下降。油门死区很大。

◦ 位置控制 **POSCTL**

- 固定翼飞行器：**Neutral inputs**可使飞行器水平飞行，如果要保持沿直线飞行，飞机将会逆风飞行。
- 多旋翼飞行器横滚控制飞机左右运动的速度，俯仰控制飞机前后运动的速度，速度都是以地面为参考系的。当横滚和俯仰的遥控杆都居中时（在死区内），飞机会稳定不动。偏航控制的是偏航角速度，这一点跟主模式相同。油门控制的是上升V下降速度，与**ALTCTL**模式中的相同。

• 自动模式 **AUTO**

◦ 悬停模式 **AUTO_LOITER**

- 固定翼飞行器：飞行器在当前位置保持当前高度（或者稍微高于当前高度，防止掉高（*good for 'i'm losing it'*））悬停，
- 多旋翼飞行器：飞行器在当前的位置、高度盘旋V悬停。

◦ 返航模式 **AUTO_RTL**

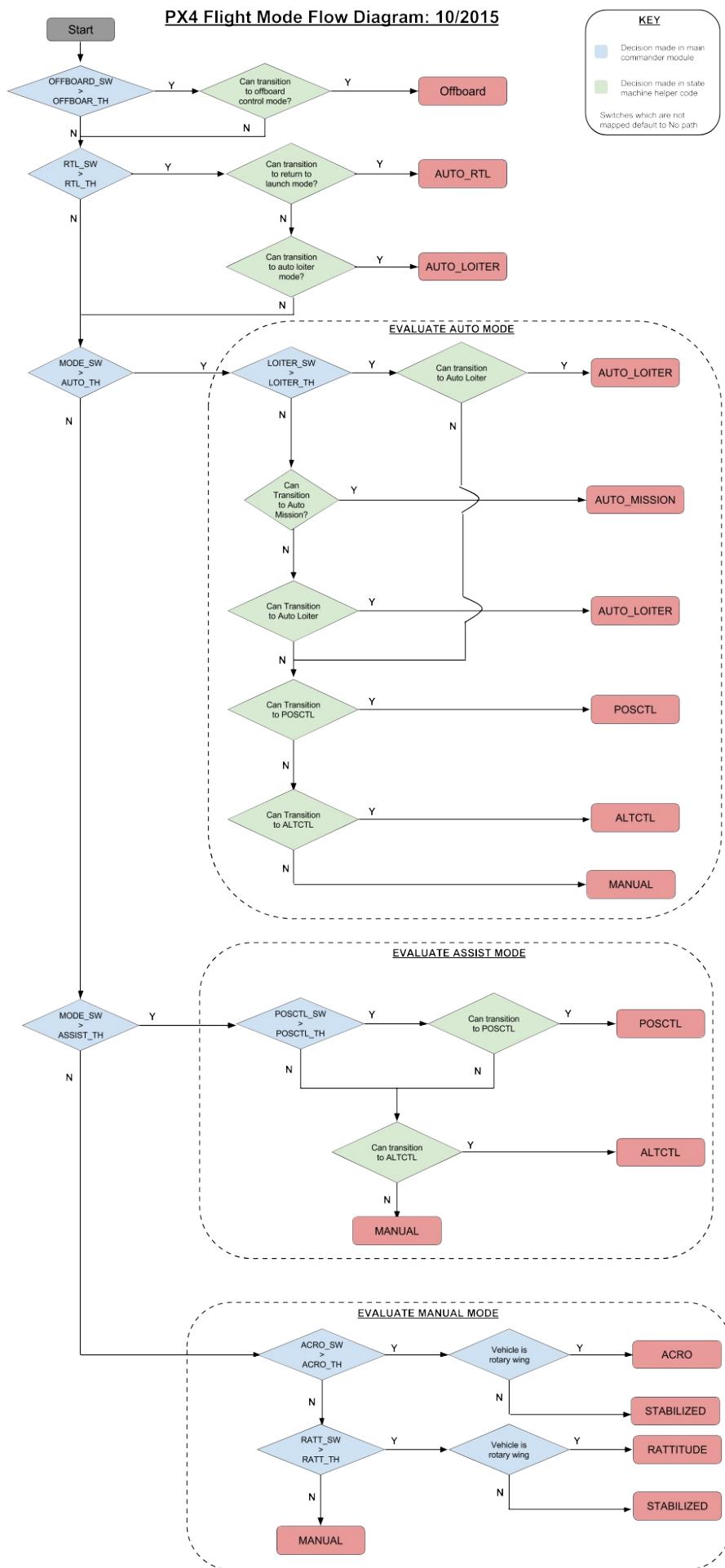
- 固定翼飞行器：飞行器返回“家”的位置然后在“家”的上方绕圈保持悬停。
- 多旋翼飞行器：如果飞机当前的高度高于起点位置的高度 + 悬停高度，则飞机将沿直线返回当前位置；如果飞机的悬停高度高于当前高度，则飞机将沿直线返回悬停位置；然后自动着陆。

◦ 任务模式 **MISSION**

- 所有系统类型：飞机按照地面控制台（**GCS**）发送的程序指令运动。如果没有接收到指令，飞机会在当前位置悬停。

- 外部控制 **OFFBOARD** 在这个模式下，飞机的位置，速度或者姿态的参考V目标V设定值由另一台通过串行线路与MAVLink连接的电脑提供。这些外部的设定值可以由MAVROS或者Dronekit这种应用程序接口提供。

飞行模式评估图



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

结构预览

官网英文原文地址：<http://dev.px4.io/concept-architecture.html>

PX4由两个层次组成：一是飞行控制栈(flight stack)，即自驾仪的软件解决方案，二是中间件，一种可以支持任意类型自主机器人的通用机器人中间件。

所有的无人机机型，事实上所有的包括船舶在内的机器人系统，都具有同一代码库。整个系统设计是反应式(reactive)的，这意味着：

- 所有的功能被划分为可替换部件
- 通过异步消息传递进行通信
- 该系统可以应对不同的工作负载

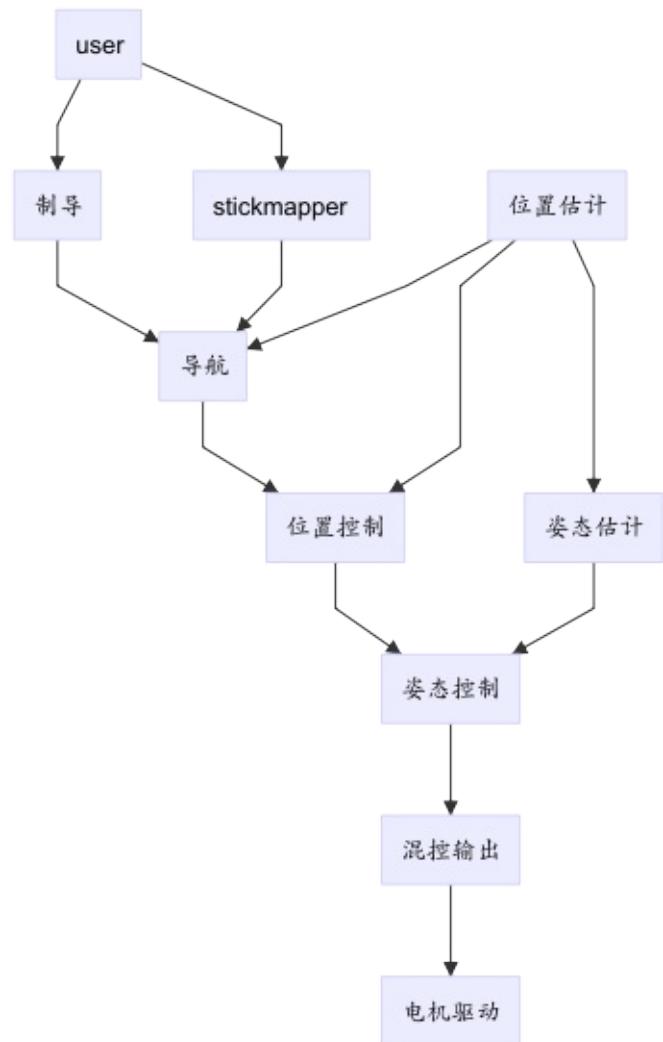
除了这些运行时的考虑外，还需要使系统各模块最大程度地可重用。

顶层软件结构

下面每一块都是单独的模块，不论是在代码，依赖性甚至是在运行时都是独立的。每个箭头是一种通过uORB进行发布/订阅调用的连接。

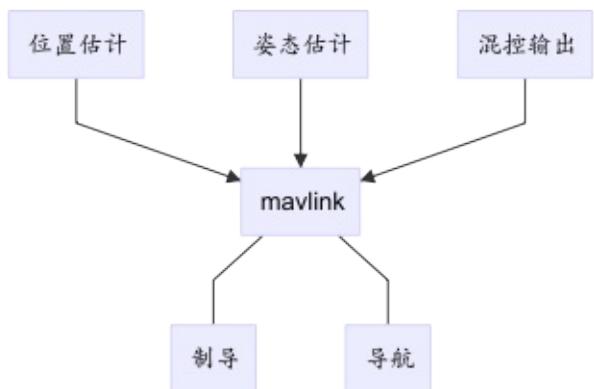
PX4结构允许其即使是在运行时，也可以快速方便地交换各个单独的模块。

控制器/混控器是针对于特定机型的（如多旋翼，垂直起降或其他飞机），但是顶层任务管理模块如控制模块，导航模块是可以在不同平台共享的。



地面站通讯架构

与地面站（GCS）之间的交互是通过一种“商业逻辑”应用程序来处理的，包括如 `commander`（一般命令与控制，例如解锁），`navigator`（接受任务并将其转为底层导航的原始数据）和 `mavlink` 应用，`mavlink` 用于接受 MAVLink 数据包并将其转换为板载 uORB 数据结构。这种隔离方式使架构更为清晰，可以避免系统对 MAVLink 过于依赖。`MAVLink` 应用也会获取大量的传感器数据和状态估计值，并将其发送到地面站。



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

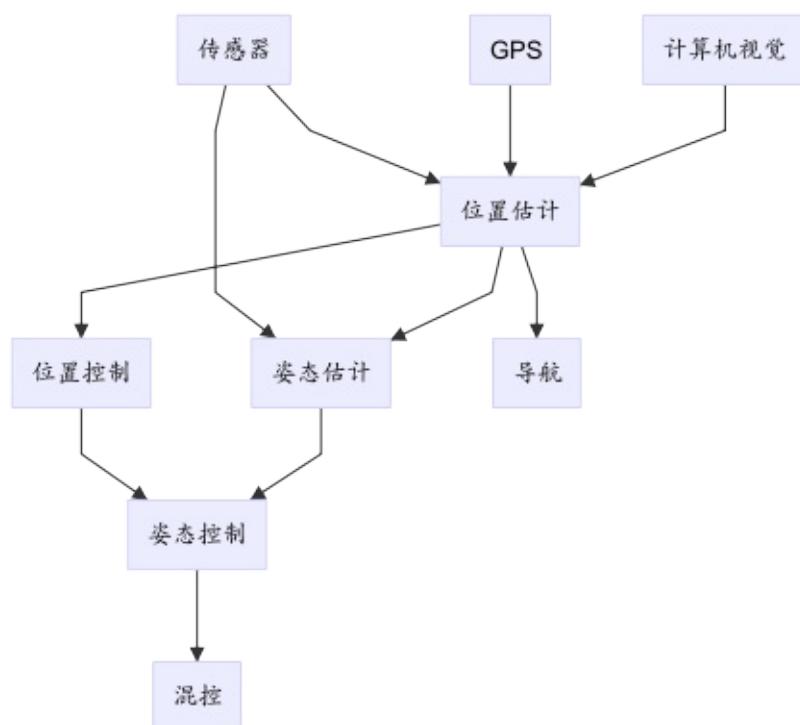
PX4飞行控制栈

官网英文原文地址：<http://dev.px4.io/concept-flight-stack.html>

PX4飞行控制栈集成了各种自主无人机的制导、导航以及控制算法。支持的机型包括固定翼，多旋翼以及垂直起降飞行器，算法包括姿态估计算法和姿态控制算法。

估计与控制结构

下图所示为一个典型框图（typical blocks）的实现示例。根据飞行器的不同，其中的一些框图（blocks）也可以组成一个单独的应用（例如，当我们需要一个特定飞行器的模型预测控制器时）



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

PX4 Middleware

官网英文原文地址：<http://dev.px4.io/concept-middleware.html>

PX4中间件主要由内置传感器的驱动和基于发布-订阅（publish-subscribe）的中间件组成，其中发布-订阅中间件用于将这些传感器与飞行控制运行的应用程序进行通讯连接。

使用发布-订阅计划意味着：

- 系统是响应式的，即当有新的有效数据时系统能够立即更新
- 系统是完全并行运行的
- 系统组件能够在线程安全的方式下从任何地方使用数据

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

混控和执行器

官网英文原文地址：<http://dev.px4.io/concept-mixing.html>

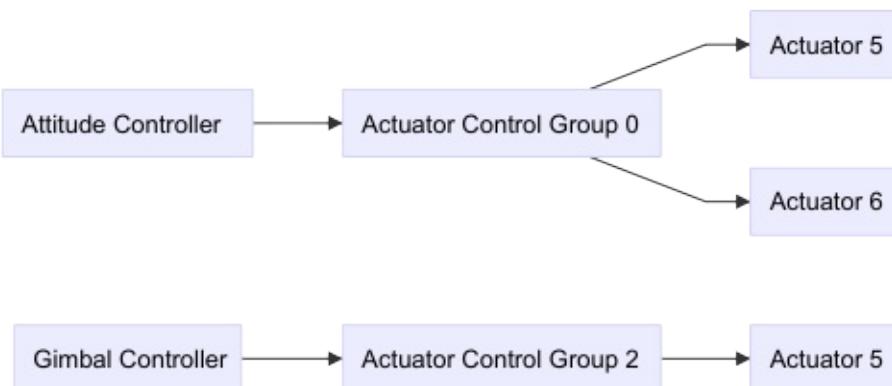
PX4架构保证了核心控制器中不需要针对机身布局做特别处理。

混控指的是把控制力指令（例如：'右转'）分配到直接控制电机以及舵机的作动器指令。对直升机而言，每个副翼由一个舵机控制，那么混控就指的是控制其中一个副翼抬起而另一个副翼落下。同样的，对多旋翼而言，俯仰操作需要改变所有电机的转速。

将混控逻辑从实际姿态控制器中分离出来可以大大提高复用性。

Control Pipeline

A particular controller sends a particular normalized force or torque demand (scaled from -1..+1) to the mixer, which then sets individual actuators accordingly. The output driver (e.g. UART, UAVCAN or PWM) then scales it to the actuators native units, e.g. a PWM value of 1300.



Control Groups

PX4 uses control groups (inputs) and output groups. Conceptually they are very simple: A control group is e.g. `attitude`, for the core flight controls, or `gimbal` for payload. An output group is one physical bus, e.g. the first 8 PWM outputs for servos. Each of these groups has 8 normalized (-1..+1) command ports, which can be mapped and scaled through the mixer. A mixer defines how each of these 8 signals of the controls are connected to the 8 outputs.

For a simple plane control 0 (roll) is connected straight to output 0 (aileron). For a multicopter things are a bit different: control 0 (roll) is connected to all four motors and combined with throttle.

Control Group #0 (Flight Control)

- 0: roll (-1..1)
- 1: pitch (-1..1)
- 2: yaw (-1..1)
- 3: throttle (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: flaps (-1..1)
- 5: spoilers (-1..1)
- 6: airbrakes (-1..1)
- 7: landing gear (-1..1)

Control Group #1 (Flight Control VTOL/Alternate)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Control Group #2 (Gimbal)

- 0: gimbal roll
- 1: gimbal pitch
- 2: gimbal yaw
- 3: gimbal shutter
- 4: reserved
- 5: reserved
- 6: reserved
- 7: reserved (parachute, -1..1)

Control Group #3 (Manual Passthrough)

- 0: RC roll
- 1: RC pitch

- 2: RC yaw
- 3: RC throttle
- 4: RC mode switch
- 5: RC aux1
- 6: RC aux2
- 7: RC aux3

Control Group #6 (First Payload)

- 0: function 0 (default: parachute)
- 1: function 1
- 2: function 2
- 3: function 3
- 4: function 4
- 5: function 5
- 6: function 6
- 7: function 7

Virtual Control Groups

These groups are NOT mixer inputs, but serve as meta-channels to feed fixed wing and multicopter controller outputs into the VTOL governor module.

Control Group #4 (Flight Control MC VIRTUAL)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

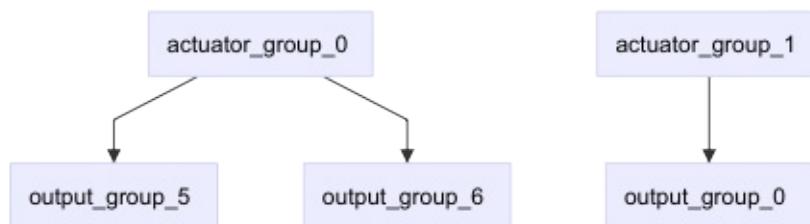
Control Group #5 (Flight Control FW VIRTUAL)

- 0: roll ALT (-1..1)
- 1: pitch ALT (-1..1)
- 2: yaw ALT (-1..1)
- 3: throttle ALT (0..1 normal range, -1..1 for variable pitch / thrust reversers)
- 4: reserved / aux0

- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

Mapping

Since there are multiple control groups (like flight controls, payload, etc.) and multiple output groups (first 8 PWM outputs, UAVCAN, etc.), one control group can send command to multiple output groups.



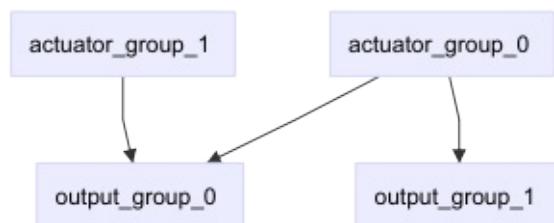
关键概念

PX4使用控制组（输入）和输出组。从概念上来说他们非常简单：对核心飞行控制来说，控制组是'姿态'，对载荷来说，则是'云台'。输出组则是一条物理总线，例如用于舵机的基本8路PWM输出。每个组有8个标准化的指令端口，这些端口可以通过混控映射和缩放。混控器定义了8路控制信号如何与8路输出信号相联系。

对简单的固定翼而言：control0（滚转）与output0（升降舵）直接相连。对多旋翼而言则不同：control0（滚转）与油门混合后与所有的电机相连。

映射

因为存在许多控制组（例如飞行控制组，载荷组等）和许多输出组（例如基本8路PWM输出组，UAVCAN组等），所以一个控制组可以向多个输出组发送指令。



PX4混控器定义

ROMFS/px4fmu_common/mixers中的文件实现了预定义机架所使用的混控器。它们可以用于自定义机架或者一般的测试。

语法

mixer通过文本文件定义；以单个大写字母加一个冒号开始的行是有效的。其它的行则会被忽略，这意味着注释可以自由地在定义中穿插使用。

每个文件可以定义多个混控器；混控器与作动器的分配关系由读取混控器定义的设备决定，作动器输出数目则由混控器决定。

例如：每个简单混控器或者空混控器按照它们在混控器文件中出现的顺序对应到输出1到输出x。

一个混控器定义通常具有如下形式：

```
<tag>: <mixer arguments>
```

tag标签决定混控器的类型；'M'对应简单求和混控器，'R'对应多旋翼混控器，等等。

空混控器

空混控器不接受控制输入并产生单个作动器输出，其输出值恒为零。空混控器的典型用法是在一组定义作动器特定输出模式的混控器组中占位。

空混控器定义形式如下：

```
Z:
```

简单混控器

简单混控器将0个或多个控制输入混合为单个作动器输出。所有输入被缩放后，经过混合函数得到混合后的输入，最后再经过输出缩放产生输出信号。

简单混控器定义如下：

```
M: <control count>
0: <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>
```

如果 `<control count>` 为0，那么混合结果实际上为0，混控器将输出一个定值，这个值是在 `<lower limit>` 和 `<upper limit>` 限制下的 `<offset>`。

第二行用前文讨论过的缩放参数定义了输出缩放器。计算以浮点操作被执行，存储在定义文件中的值经过了因子10000的缩放，即偏移量-0.5会被存储为-5000。

紧跟在 `<control count>` 词目之后的定义描述了控制输入以及它们的缩放，形式如下：

```
S: <group> <index> <-ve scale> <+ve scale> <offset> <lower limit> <upper limit>
```

`<group>` 值标示了控制输入来源，缩放器从中读取控制量，`<index>` 值则是控制量在组内的序号。这些值对读取混控器定义的设备而言都是特定的。

当用来混合载体控制时，控制组0是载体姿态控制组，序号0到3通常对应滚转，俯仰，偏航和油门。

混控器定义行中剩下的域则用来配置缩放器，参数如前文讨论。计算以浮点操作被执行，存储在定义文件中的值经过了因子10000的缩放，即偏移量-0.5会被存储为-5000。

多旋翼混控器

多旋翼混控器将4个控制输入（滚转，俯仰，偏航，油门）混合至一组作动器输出，这些作动器用来驱动电机转速控制器。

多旋翼混控器定义如下所示：

```
R: <geometry> <roll scale> <pitch scale> <yaw scale> <deadband>
```

支持的构型包括：

- 4x - X型布局四旋翼
- 4+ - +型布局四旋翼
- 6x - X型布局六旋翼
- 6+ - +型布局六旋翼
- 8x - X型布局八旋翼
- 8+ - +型布局八旋翼

每个滚转，俯仰，偏航缩放值定义了滚转，俯仰，偏航控制相对于油门控制的缩放。计算以浮点操作被执行，存储在定义文件中的值经过了因子10000的缩放，即偏移量-0.5会被存储为-5000。

滚转，俯仰和偏航输入的范围为-1.0到1.0，而油门输入的范围为0.0到1.0，作动器输出范围为-1.0到1.0。

当某个作动器饱和时，为保证该作动器值不超出范围，所有的作动器值都会被重新缩放。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

PWM_limit状态机

官网英文原文地址：http://dev.px4.io/concept-pwm_limit.html

PWM_limit 状态机根据解锁前和解锁后的输入控制PWM输出，并提供'armed'置1和解锁信号生效之间的延迟来延缓油门上升。

快速概要

输入

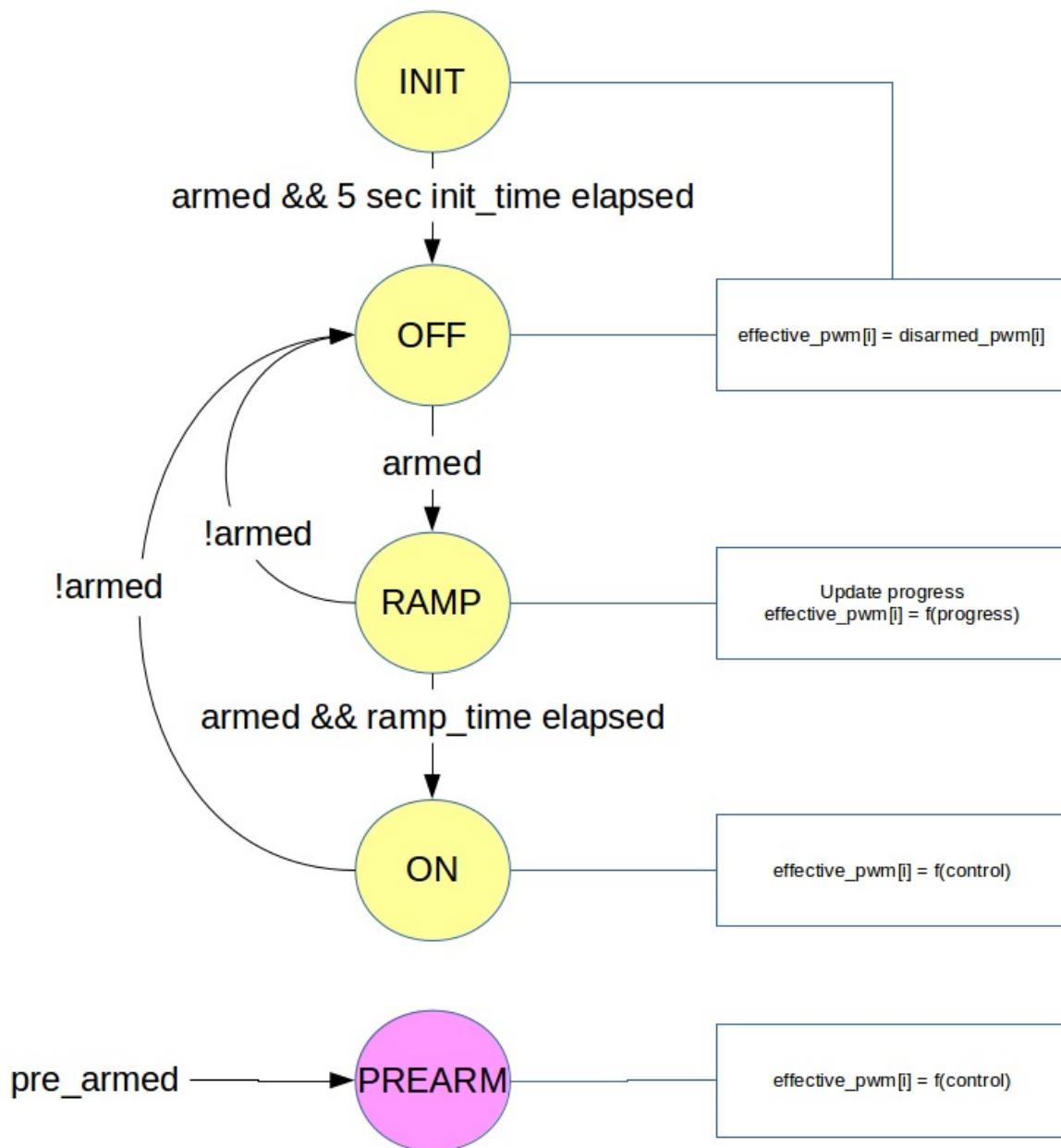
- **armed**: 置1使能诸如旋转螺旋桨的危险行为。
- **pre-armed**: 置1使能诸如移动控制面的良性行为。
 - 这个输入覆盖当前状态。
 - pre-arm置1无视当前状态，立即强制转移到状态ON，值0则回复到当前状态。

状态

- INIT和OFF
 - pwm输出值设定为未解锁值。
- RAMP
 - pwm输出值从未解锁值上升到最小值。
- ON
 - pwm输出值根据控制量设定。

状态转移图

PWM_LIMIT_STATE_



Note: input pre_armed overrides the current state while asserted. When deasserted the state machine functions normally. Throttle controls must be set to NaN while pre_armed is asserted.

教程

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

QGroundControl

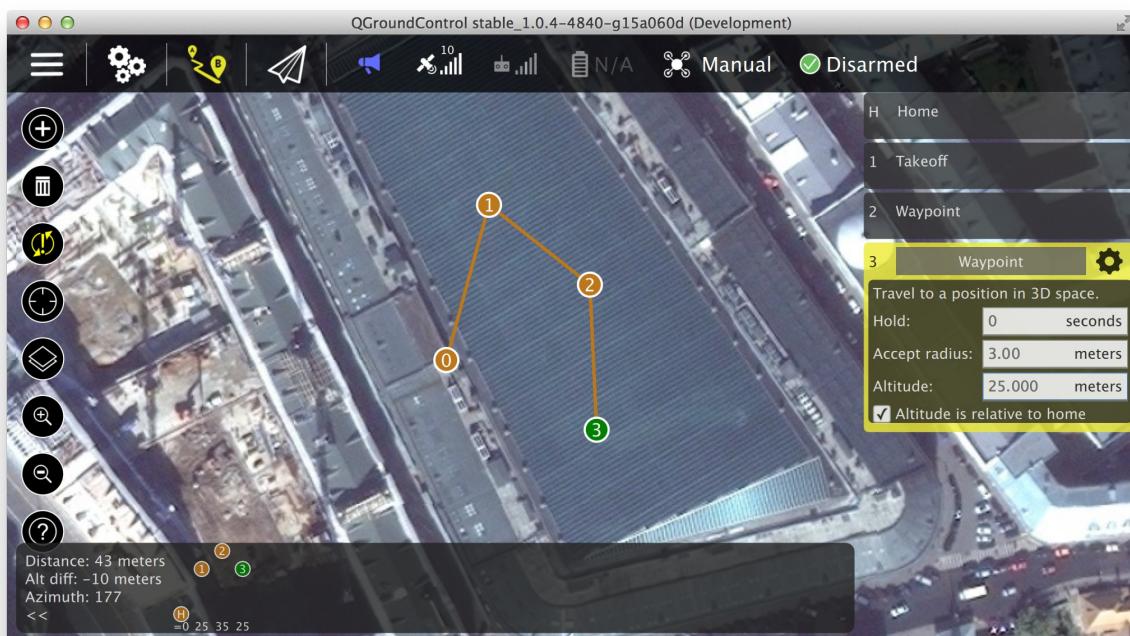
官网英文原文地址：<http://dev.px4.io/qgroundcontrol-intro.html>

QGroundControl是一个基于PX4自动驾驶仪配置和飞行的应用程序。并且跨平台支持所有的主流操作系统：

- 手机系统: Android 和 iOS (目前专注于平板电脑)
- 桌面系统: Windows, Linux, Mac OS

任务规划

规划一个新的任务，切换到任务菜单，点击左上角的“+”图标然后在地图上单击创建出一个任务点。同时在旁边将打开一个快捷菜单用来调整任务点。点击高亮的任务发送图标把任务信息发送到驾驶仪设备。



任务飞行

切换到飞行菜单。让地图上的任务路径保持可见。点击当前的飞行模式使其变为任务模式并且点击锁定按钮来解锁飞行器。如果飞行器已经飞行在空中它将直接飞向第一个任务点并且跟随任务路径飞行。。



参数设置

切换到设置菜单。滚动左边的菜单到最底部并且点击参数图标。可以通过双击某一项参数来弹出一个带有详细信息描述的可编辑菜单来改变参数。

Multicopter Attitude Control Parameters		
MC_ACRO_P_MAX	360.000 deg/s	Max acro pitch rate
MC_ACRO_R_MAX	360.000 deg/s	Max acro roll rate
MC_ACRO_Y_MAX	360.000 deg/s	Max acro yaw rate
MC_PITCHRATE_D	0.003	Pitch rate D gain
MC_PITCHRATE_FF	0.000	Pitch rate feedforward
MC_PITCHRATE_I	0.050	Pitch rate I gain
MC_PITCHRATE_MAX	220.000 deg/s	Max pitch rate
MC_PITCHRATE_P	0.150	Pitch rate P gain
MC_PITCH_P	7.000 1/s	Pitch P gain
MC_RATT_TH	1.000	Threshold for Rattitude mode
MC_ROLLRATE_D	0.003	Roll rate D gain
MC_ROLLRATE_FF	0.000	Roll rate feedforward
MC_ROLLRATE_I	0.050	Roll rate I gain
MC_ROLLRATE_MAX	220.000 deg/s	Max roll rate
MC_ROLLRATE_P	0.150	Roll rate P gain
MC_ROLL_P	7.000	Roll P gain

安装

QGroundControl可以从[这里下载 QGroundControl.](#)

开发人员建议使用最新的版本，而不是稳定版本。

从源代码编译

鼓励固件开发人员使用源代码来编译匹配他们飞行代码的版本。

查看[QGroundControl编译说明](#) 来学习安装Qt以及编译源代码。

© PX4WIKI team all right reserved , powered by Gitbook该文件修订时间：2017-01-22

12:56:03

第一个应用程序教程(Hello Sky)

官网英文原文地址：<http://dev.px4.io/tutorial-hello-sky.html>

本教程详细解释了如何创建一个新的板载应用程序，以及如何运行它。

前提条件

- Pixhawk 或者 Snapdragon兼容的自驾仪
- 安装PX4 工具链
- Github 账号 ([免费注册](#))

第一步：文件设置

为了更方便的管理你的个人代码和上传更新到原始代码仓库，强烈建议使用fork命令通过GIT版本控制系统得到一个新的PX4固件。

- 在[github上注册](#)一个账户；
- 登陆[px4的固件托管网址](#)，然后点击右上方的**FORK**按钮；
- 点击到你fork后的站点，复制你的私人PX4代码仓库的URL地址。
- 克隆你的代码仓库到你的硬盘中，在命令行中输入：`git clone https://github.com/<youraccountname>/Firmware.git` 对于windows用户，请参考[github帮助](#)中的介绍。例如在github创建的官方应用程序中使用fork/clone命令克隆仓库到本地硬盘中。
- 更新px4代码包含的git子模块：运行命令行工具（在windows上运行PX4终端）

```
cd Firmware
git submodule init
git submodule update --recursive
```

打开你本地硬盘克隆的软件仓库的 `Firmware/src/examples/` 文件夹，查看里面的文件。

第二步：创建小型应用程序

在 `Firmware/src/examples/px4_simple_app` 下，创建一个新的C语言文件 `px4_simple_app.c`。（该文件已存在，你可以直接删掉它，来完全自主编辑学习。）

从默认头文件和main主函数开始，编辑这个文件。

注意这个文件中的代码风格，所有PX4的软件版本都将遵守这个风格。

```
/****************************************************************************
 *
 * Copyright (c) 2012-2015 PX4 Development Team. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name PX4 nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 ****/

```

```
/**
 * @file px4_simple_app.c
 * Minimal application example for PX4 autopilot
 *
 * @author Example User <mail@example.com>
 */

#include <px4_config.h>
#include <px4_tasks.h>
#include <px4_posix.h>
#include <unistd.h>
#include <stdio.h>
#include <poll.h>
#include <string.h>

#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>
```

```
#include <uORB/topics/vehicle_attitude.h>

__EXPORT int px4_simple_app_main(int argc, char *argv[]);

int px4_simple_app_main(int argc, char *argv[])
{
    PX4_INFO("Hello Sky!");
    return OK;
}
```

第三步：在NuttShell中注册应用程序，然后编译

现在应用程序已经完成并且能够运行，但还没有注册成NuttShell命令行工具。如果想要将应用程序编译到固件中，将它加到下面的模块编译列表中：

- Pixhawk v1/2: [Firmware/cmake/configs/nuttx_px4fmu-v2_default.cmake](#)
- Pixracer: [Firmware/cmake/configs/nuttx_px4fmu-v4_default.cmake](#)

在你的应用程序的下面的文件中添加一行：

```
examples/px4_simple_app
```

编译它：

- Pixhawk v1/2: `make px4fmu-v2_default`
- Pixhawk v3: `make px4fmu-v4_default`

第四步：上传并且测试应用程序

使能uploader然后重置开发板：

- Pixhawk v1/2: `make px4fmu-v2_default upload`
- Pixhawk v3: `make px4fmu-v4_default upload`

在你重置开发板之前，会在后面打印如下的编译信息：

```
Loaded firmware for X,X, waiting for the bootloader...
```

一旦开发板重置成功并且应用程序上传成功，打印信息：

```
Erase : [=====] 100.0%
Program: [=====] 100.0%
Verify : [=====] 100.0%
Rebooting.

[100%] Built target upload
```

连接到终端

现在通过串口或USB连接到[系统命令行终端](#)（新手玩家第一次进行USB或者串口连接系统命令行终端，请点开上面的链接，按照要求进行系统控制台安装，安装完毕后，输入回车键打开终端（linux）或者命令行工具（windows））。点击回车键能打开shell命令行工具：

```
nsh>
```

输入“`help`”然后回车

```
nsh> help
  help usage: help [-v] [<cmd>]

[      df      kill      mkfifo      ps      sleep
?      echo     losetup    mkrd      pwd      test
cat    exec     ls        mh        rm       umount
cd     exit     mb        mount    rmdir    unset
cp     free     mkdir     mv        set      usleep
dd     help     mkfatfs  mw        sh       xd

Builtin Apps:
reboot
perf
top
..
px4_simple_app
..
sercon
serdis
```

现在‘`px4_simple_app`’已经是一个可用的命令了。输入 `px4_simple_app` 命令然后回车：

```
nsh> px4_simple_app
Hello Sky!
```

现在应用程序已经成功注册到系统中，能够被扩展并且运行一些有用的任务。

第五步：读取传感器数据

为了实现一些功能，应用程序需要读取传感器的输入然后反应到对电机或者舵机的输出中。注意在这里，PX平台真正的硬件抽象的概念在这里体现---当硬件平台或者传感器更新，完全不需要更新你的应用程序或者更新传感器驱动程序。

在PX4中，应用程序间发送的单独的消息叫做“topics”，在本教程中，我们关心的话题是“多传感器间的uORB消息机制”（[sensor_combined topic](#)）。这些消息机制使得整个系统能够同步传感器数据。

订阅一个话题是非常迅速并且简洁的：

```
#include <uORB/topics/sensor_combined.h>
..
int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
```

“sensor_sub_fd”是一个文件描述符，可以用来非常高效地实现对新数据的阻塞式等待。当前线程进入休眠状态，当新数据可用时，它自动地被调度程序唤醒，因此在数据等待时，不会占用任何CPU资源。为了实现这个功能，我们使用poll()函数

[<http://pubs.opengroup.org/onlinepubs/007908799/xsh/poll.html>]，即POSIX系统调用。

在消息读取中加入“poll()”机制：

```
#include <poll.h>
#include <uORB/topics/sensor_combined.h>
..
int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));

/* one could wait for multiple topics with this technique, just using one here */
px4_pollfd_struct_t fds[] = {
    { .fd = sensor_sub_fd, .events = POLLIN },
};

while (true) {
    /* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
    int poll_ret = px4_poll(fds, 1, 1000);
    ..

    if (fds[0].revents & POLLIN) {
        /* obtained data for the first file descriptor */
        struct sensor_combined_s raw;
        /* copy sensors raw data into local buffer */
        orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
        printf("[px4_simple_app] Accelerometer:\t%8.4f\t%8.4f\t%8.4f\n",
               (double)raw.accelerometer_m_s2[0],
               (double)raw.accelerometer_m_s2[1],
               (double)raw.accelerometer_m_s2[2]);
    }
}
```

编译应用程序：

```
make
```

第六步：测试uORB消息读取机制

最后一步，开始你的应用程序，并且切换到后台应用。

```
px4_simple_app &
```

你的应用程序会向串口输出当前传感器的值：

```
[px4_simple_app] Accelerometer: 0.0483      0.0821      0.0332
[px4_simple_app] Accelerometer: 0.0486      0.0820      0.0336
[px4_simple_app] Accelerometer: 0.0487      0.0819      0.0327
[px4_simple_app] Accelerometer: 0.0482      0.0818      0.0323
[px4_simple_app] Accelerometer: 0.0482      0.0827      0.0331
[px4_simple_app] Accelerometer: 0.0489      0.0804      0.0328
```

它会在输出5次数据后退出。下一篇教程中会介绍如何编写一个能通过命令行控制的后台应用。

第七部：打印数据

为了能获取到计算后的数据，下一步就是“打印”这些结果。如果我们知道某一个消息是使用mavlink协议转发给地面控制站的，我们就可以通过这个消息去查看结果。例如我们通过这个方法来获得高度信息的消息。

接口非常简单：初始化消息的结构体，然后公告这条消息：

```
#include <uORB/topics/vehicle_attitude.h>
..
/* advertise attitude topic */
struct vehicle_attitude_s att;
memset(&att, 0, sizeof(att));
orb_advert_t att_pub_fd = orb_advertise(ORB_ID(vehicle_attitude), &att);
```

在主循环中，当消息准备好时，打印这条消息。

```
orb_publish(ORB_ID(vehicle_attitude), att_pub_fd, &att);
```

修改后的完整的示例代码如下：

```
#include <px4_config.h>
#include <px4_tasks.h>
#include <px4_posix.h>
#include <unistd.h>
```

```

#include <stdio.h>
#include <poll.h>
#include <string.h>

#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>
#include <uORB/topics/vehicle_attitude.h>

__EXPORT int px4_simple_app_main(int argc, char *argv[]);

int px4_simple_app_main(int argc, char *argv[])
{
    PX4_INFO("Hello Sky!");

    /* subscribe to sensor_combined topic */
    int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
    orb_set_interval(sensor_sub_fd, 1000);

    /* advertise attitude topic */
    struct vehicle_attitude_s att;
    memset(&att, 0, sizeof(att));
    orb_advert_t att_pub = orb_advertise(ORB_ID(vehicle_attitude), &att);

    /* one could wait for multiple topics with this technique, just using one here */
    px4_pollfd_struct_t fds[] = {
        { .fd = sensor_sub_fd, .events = POLLIN },
        /* there could be more file descriptors here, in the form like:
         * { .fd = other_sub_fd, .events = POLLIN },
         */
    };

    int error_counter = 0;

    for (int i = 0; i < 5; i++) {
        /* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
        int poll_ret = px4_poll(fds, 1, 1000);

        /* handle the poll result */
        if (poll_ret == 0) {
            /* this means none of our providers is giving us data */
            PX4_ERR("[px4_simple_app] Got no data within a second");
        }

        } else if (poll_ret < 0) {
            /* this is seriously bad - should be an emergency */
            if (error_counter < 10 || error_counter % 50 == 0) {
                /* use a counter to prevent flooding (and slowing us down) */
                PX4_ERR("[px4_simple_app] ERROR return value from poll(): %d"
                       , poll_ret);
            }
        }

        error_counter++;

    } else {

```

```

    if (fds[0].revents & POLLIN) {
        /* obtained data for the first file descriptor */
        struct sensor_combined_s raw;
        /* copy sensors raw data into local buffer */
        orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
        PX4_WARN("[px4_simple_app] Accelerometer:\t%8.4f\t%8.4f\t%8.4f",
                  (double)raw.accelerometer_m_s2[0],
                  (double)raw.accelerometer_m_s2[1],
                  (double)raw.accelerometer_m_s2[2]);

        /* set att and publish this information for other apps */
        att.roll = raw.accelerometer_m_s2[0];
        att.pitch = raw.accelerometer_m_s2[1];
        att.yaw = raw.accelerometer_m_s2[2];
        orb_publish(ORB_ID(vehicle_attitude), att_pub, &att);
    }

    /* there could be more file descriptors here, in the form like:
     * if (fds[1..n].revents & POLLIN) {} */
    }
}

PX4_INFO("exiting");

return 0;
}

```

第八步：运行整个示例

运行你的应用程序：

```
px4_simple_app
```

如果打开QGroundControl，你就能通过实时绘图程序(Tools -> Analyze)来获得实时的传感器数据。 If you start QGroundControl, you can check the sensor values in the realtime plot (Tools -> Analyze)

小结

这个教程包含了所有开发一个PX4应用程序需要的东西。关于uORB消息机制的详细信息参见可以从[这里](#)获得。所有的头文件都已经良好的注释作为参考。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

QGC的视频流

官网英文原文地址：<http://dev.px4.io/advanced-videostreaming-qgc.html>

这页面说明如何建立一台带一个镜头(Logitech C920) 的配套工业计算机Odroid C1 ，视频流通过Odroid C1 (ODROID C1) 传输到网络计算机和在计算机上的QDC应用显示运行。

整个硬件设置如下所示。它由以下几个部分组成：

- Odroid C1
- 罗技镜头 C920
- WIFI猫 TP-LINK TL-WN722N

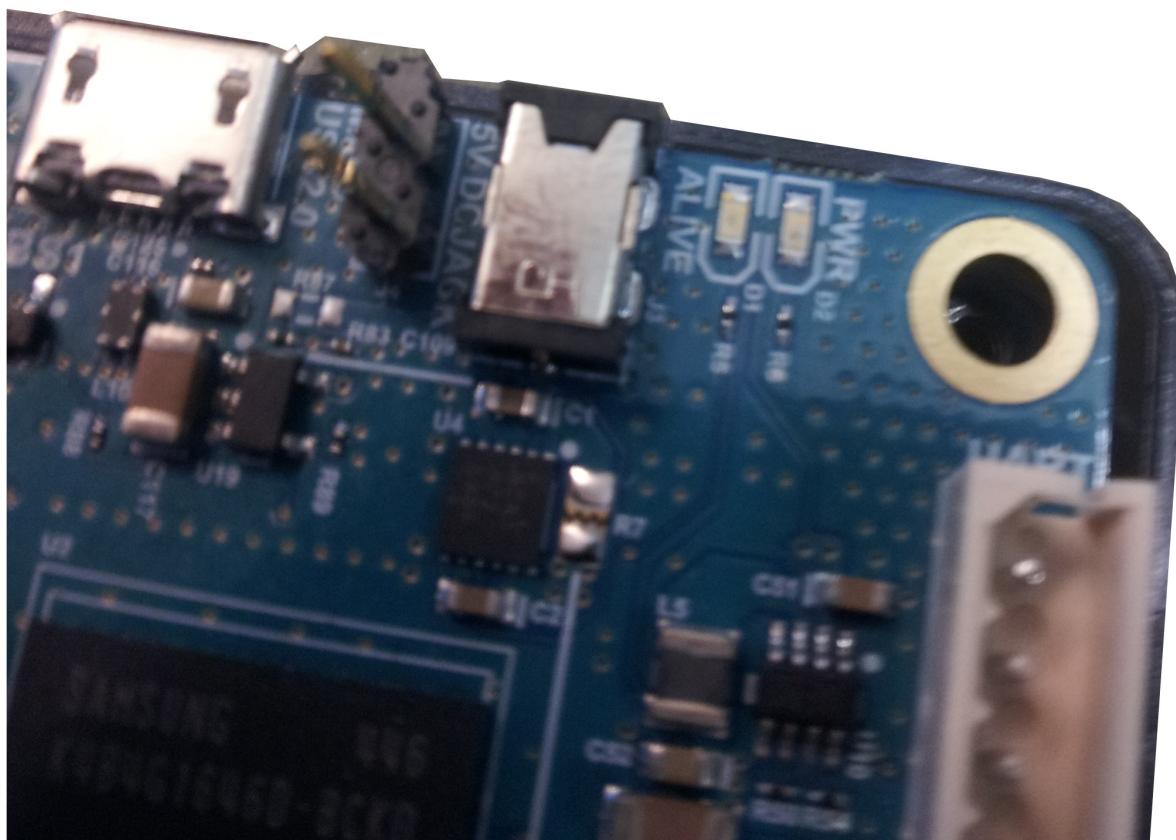


安装Odroid C1 Linux的环境

按照Odroid C1安装教程，安装Ubuntu 14.04)的运行环境，在本教程中 [Odroid C1 tutorial](#). 还说明如何用串口电缆接入Odroid C1 (ODROID C1) 以及如何建立以太网连接。

建立连接替代电源

Odroid C1 能够通过5V DC电源插孔。如果ODROID是安装在无人机的，建议采用焊接两引脚通孔焊接5V直流插孔固定的方法如下图所示。[method](#)。电源的使用是连接到通过跳线(红色如上面的图片)的ODROID C1 的直流电压 (5 V) 和接地电路连接在示例设置中Odroid C1 的跳线地线针脚上 (如上图黑线)。



启用ODROID C1 WiFi连接

在这本教程的WiFi模块采用TP-LINK tl-wn722n。要启用的ODROID C1的WiFi连接，按照ODROID C1教程的步骤描述用WiFi天线建立网络连接。[Odroid C1 tutorial in](#)

配置 WiFi 接入点

本节说明如何设置ODROID C1一个数据接入点。内容在原有[教程](#)上作一些小的改进。启动视频流从摄像头的通过到地面站，运行在计算机则不需要遵循本节。然而，这里表示是因为建立ODROID C1像允许以一个独立的方式使用系统的数据接入点。TP-LINK TL-WN722N使用像WiFi模块。在随后的步骤它是假定指定名称的ODROID系列C1 WLAN 0到你的WiFi模块。改变WLAN 0的所有事件如果相应接口不同（如wlan1）。

机载计算机作为接入点

在深入的解释更多，你可以看看[RPI-Wireless-Hotspot](#)

安装必要的软件

```
sudo apt-get install hostapd udhcpd
```

配置 DHCP. 编辑文件 `/etc/udhcpd.conf`

```
start 192.168.2.100 # This is the range of IPs that the hotspot will give to client devices
end 192.168.2.200
interface wlan0 # The device uDHCP listens on.
remaining yes
opt dns 8.8.8.8 4.2.2.2 # The DNS servers client devices will use (if routing through the
opt subnet 255.255.255.0
opt router 192.168.2.1 # The Onboard Computer's IP address on wlan0 which we will set up
opt lease 864000 # 10 day DHCP lease time in seconds
```

所有其他的“选择”条目应该被禁用或如果你知道你在做如何配置正确。

编辑文件 `/etc/default/udhcpd` 和修改行:

```
DHCPD_ENABLED="no"
```

为

```
#DHCPD_ENABLED="no"
```

您需要给机载计算机一个静态的IP地址 编辑文件 `/etc/network/interfaces` 和代替行 `iface wlan0 inet dhcp` (or `iface wlan0 inet manual`) 为:

```
auto wlan0
iface wlan0 inet static
address 192.168.2.1
netmask 255.255.255.0
network 192.168.2.0
broadcast 192.168.2.255
wireless-power off
```

停用原有（无线客户端）自动配置。修改行 (they probably will not be all next to each other or may not even be there at all):

```
allow-hotplug wlan0
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

为：

```
#allow-hotplug wlan0
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp
```

如果你遵循 [Odroid C1 教程](#) 建立无线网络连接，您已经创建的文件

`/etc/network/intefaces.d/wlan0` . 请注释在该文件中的所有行，使得这些配置不再有任何效果。

配置 HostAPD: 创建 WPA-secured 网络, 编辑文件 `/etc/hostapd/hostapd.conf` (如果它不存在就新创建) 和 加上跟随行:

```
auth_algs=1
channel=6          # Channel to use
hw_mode=g
ieee80211n=1      # 802.11n assuming your device supports it
ignore_broadcast_ssid=0
interface=wlan0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
# Change the to the proper driver
driver=nl80211
# Change these to something else if you want
ssid=OdroidC1
wpa_passphrase=QGroundControl
```

改变 `ssid=`, `channel=`, 和 `wpa_passphrase=` 你选的值. **SSID** 是传播到其他设配热点名, 通道是热点运行在什么频率, **wpa_passphrase**是无线网络密码。为更多的选项看到文件。
`/usr/share/doc/hostapd/examples/hostapd.conf.gz`. 寻找一个在这个区域没有使用的通道, 你可以使用的工具如**wavemon**。

E编辑文件 `/etc/default/hostapd` 修改行:

```
#DAEMON_CONF=""
```

为:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

你的机载电脑现在应该是主持一个无线热点。要获得启动启动的热点，运行这些额外的命令：

```
sudo update-rc.d hostapd enable  
sudo update-rc.d udhcpcd enable
```

作为一个数据接入点要有足够同时接入机载计算机和允许你的地面站连接. 如果你真的想让它作为一个真正的接入点 (WiFi机载计算机的以太网连接路由的流), 我们需要配置路由和网络地址翻译（NAT）。启用内核中的IP转发：

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

启动内核，运行下面命令:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE  
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT  
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

调整参数，运行下面命令:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

编辑文件 `/etc/network/interfaces` 和在文件最下方加上这一行:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

gstreamer 安装

在电脑上和Odroid C1 安装 gstreamer 包 和 启动流, 按照指令获取 [QGroundControl README](#).

如果你不能启动Odroid与uvch264s插件, 也可以尝试与v4l2src插件启动:

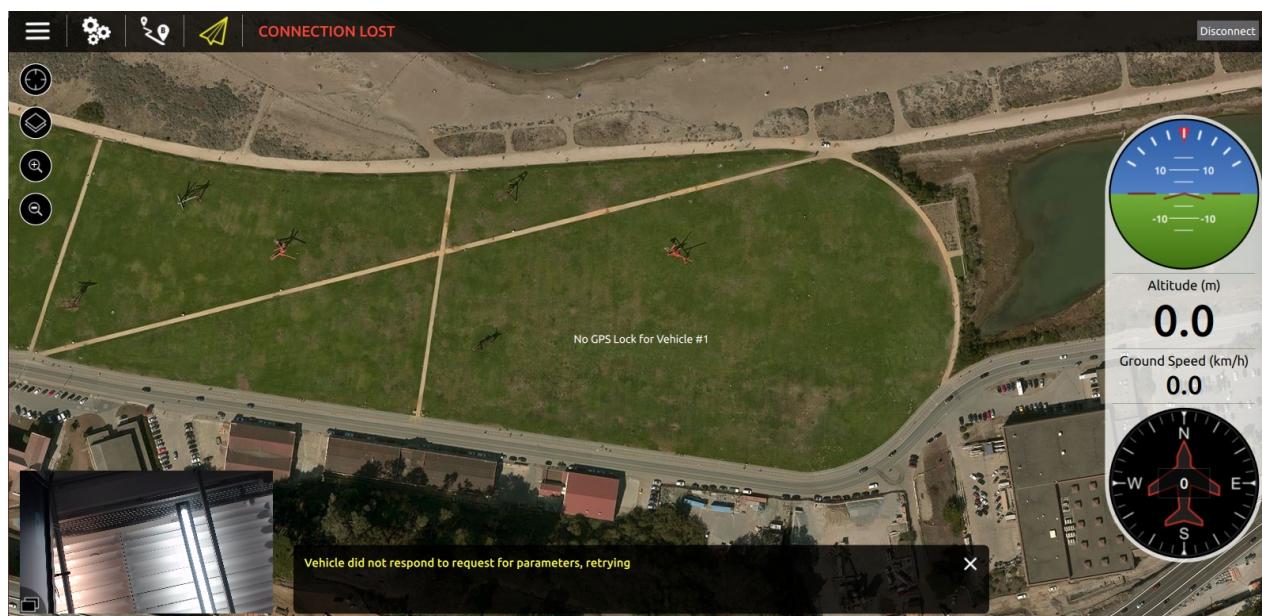
```
gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-h264,width=1920,height=1080,framerate=30/1
```





当 xxx.xxx.xxx.xxx QGC地面站IP地址正在运行, 如果看到系统错误 Permission denied , 你可以下面这个命令 sudo 。

如果一切正常, 你应该看到在底部的左上角OGC视频流在飞行模式下screeenshot显示窗口。



如果你点击了视频流, 卫星地图将在整个背景左下角的显示和视频显示。

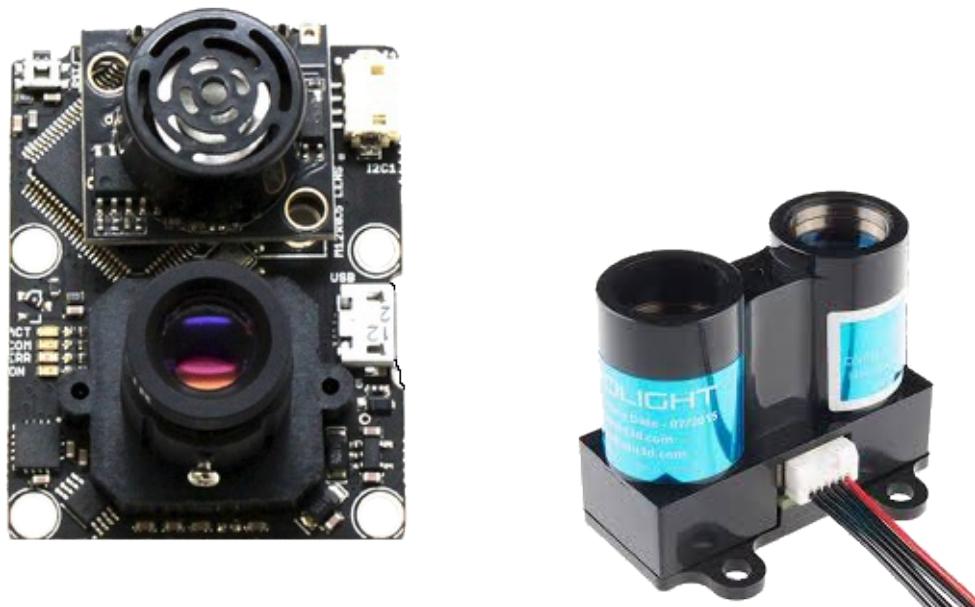
© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

光流传感器和激光雷达传感器

官网英文原文地址：http://dev.px4.io/flow_lidar_setup.html

本页展示了如何在INAV位置估计中设置PX4FLOW光流传感器和LIDAR-Liter激光雷达传感器。如下是一个定位的验证小视频。

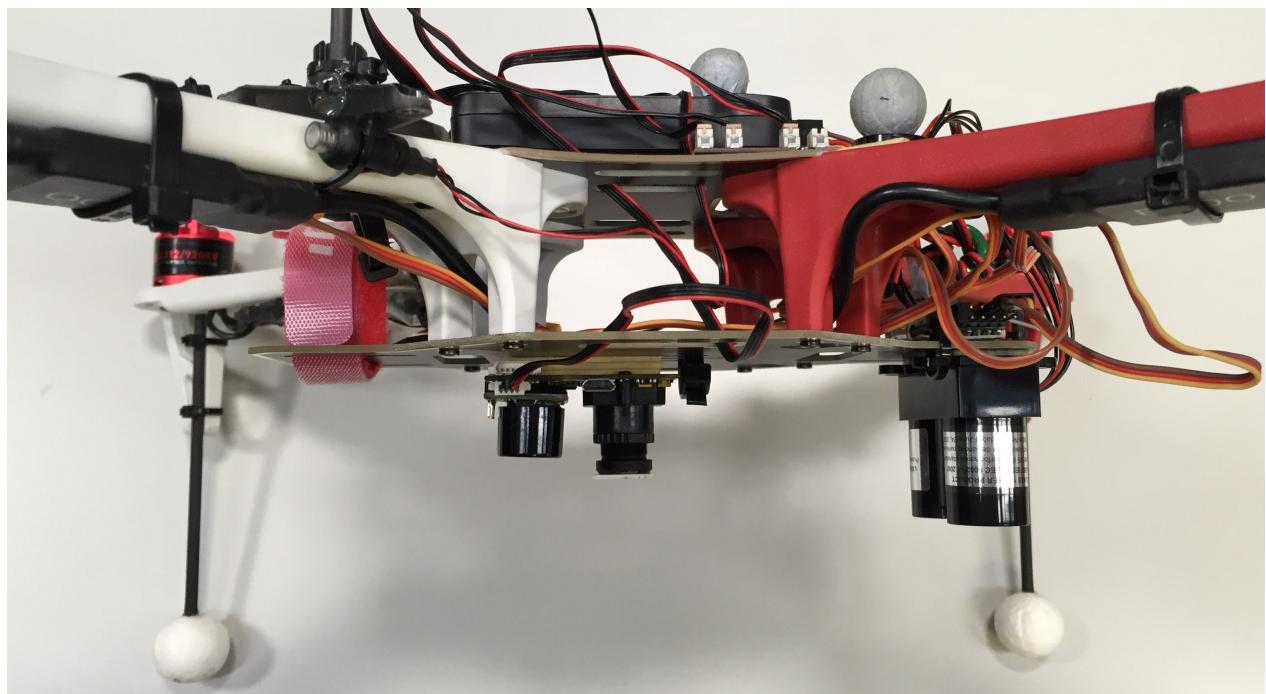
- 室内
- 室外



硬件

PX4Flow光流传感器必须指向地面，并且可以使用I2C口连接到pixhawk。

关于LIDAR-Lite激光雷达传感器的连接，请查看页面[this](#). 为了得到最好的性能，请保持PX4Flow光流传感器在一个良好的位置并且不要有太大的震动。（最好在四旋翼飞行器的下方且朝向地面）

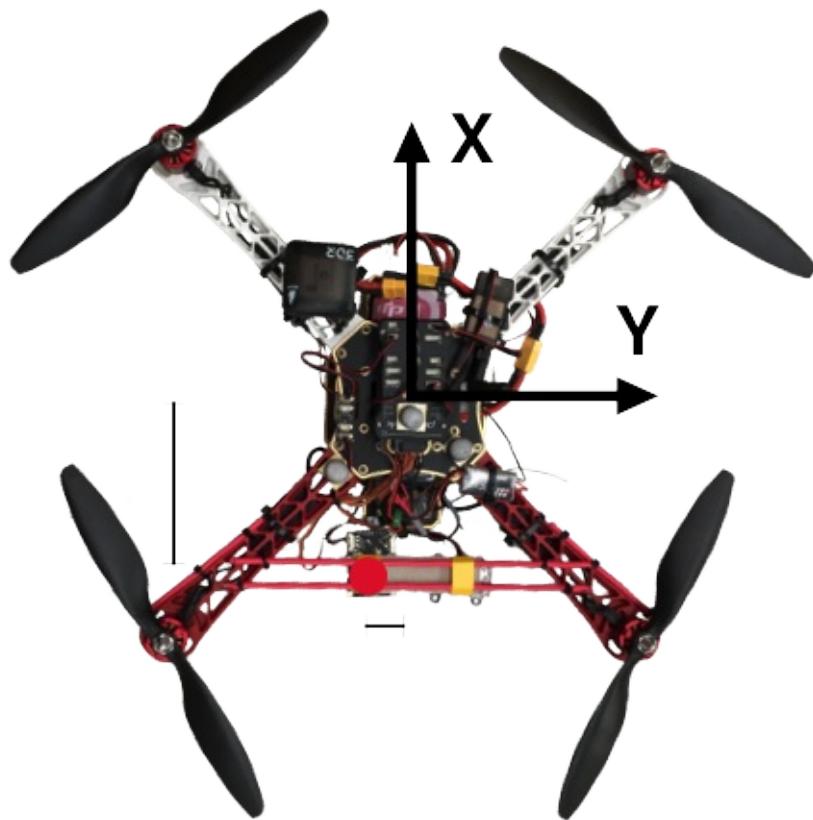


参数

所有的参数都能通过QGroundControl地面站进行修改

- `SENS_EN_LL40LS` 设置为1来启用激光雷达距离测量
- `INAV_LIDAR_EST` 设置为1来启用基于距离测量的高度估计。

- `INAV_FLOW_DIST_X` and `INAV_FLOW_DIST_Y` 这两个参数（单位：米）被用来作偏航角度的补偿。补偿的偏移量参考下面的图例。



- 上面的例子中，光流传感器的安装位置会产生一个正向的X轴的偏移和一个正向的Y轴的偏移。
- `INAV_LIDAR_OFF` 是指激光雷达高度的一个振动偏移量（单位/米），这个数值会被加入到激光定高传感器的距离测量中。

高级用户

对于高级用户，下面的参数也可以进行修改。但在你了解它们的用法前，不要修改它们。

- `INAV_FLOW_W` 是指光流的位置估计对整个姿态估计的权重。
- `INAV_LIDAR_ERR` 设置高度估计的门限值，如果测量结果值高于这个值，测量结果会被舍弃。

综合测试

官网英文原文地址：<http://dev.px4.io/tutorial-integration-testing.html>

这是综合测试，测试会自动执行([Jenkins CI](#))。

ROS / MAVROS 测试

前提：

- SITL仿真
- Gazebo
- ROS and MAVROS

执行测试

运行完整的MAVROS测试套件：

```
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
rostest px4 mavros_posix_tests_iris.launch
```

或者使用GUI查看：

```
rostest px4 mavros_posix_tests_iris.launch gui:=true headless:=false
```

编写新的MAVROS测试 (Python)

目前处于早期阶段，采用了很多简化测试(helper classes/methods etc.)。

1.) 创建新的测试脚本

测试脚本位于 `integrationtests/python_src/px4_it/mavros/`，可以参考这些脚本文件。或者查阅ROS官方文档学习如何使用[unittest](#)。

空的测试框架：

```

#!/usr/bin/env python
# [... LICENSE ...]

#
# @author Example Author <author@example.com>
#
PKG = 'px4'

import unittest
import rospy
import rosbag

from sensor_msgs.msg import NavSatFix

class MavrosNewTest(unittest.TestCase):
    """
    Test description
    """

    def setUp(self):
        rospy.init_node('test_node', anonymous=True)
        rospy.wait_for_service('mavros/cmd/arm', 30)

        rospy.Subscriber("mavros/global_position/global", NavSatFix, self.global_position)
        self.rate = rospy.Rate(10) # 10hz
        self.has_global_pos = False

    def tearDown(self):
        pass

    #
    # General callback functions used in tests
    #
    def global_position_callback(self, data):
        self.has_global_pos = True

    def test_method(self):
        """Test method description"""

        # FIXME: hack to wait for simulation to be ready
        while not self.has_global_pos:
            self.rate.sleep()

        # TODO: execute test

    if __name__ == '__main__':
        import rostest
        rostest.rosrun(PKG, 'mavros_new_test', MavrosNewTest)

```

2.) 只运行新的测试

```
# Start simulation
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
roslaunch px4 mavros_posix_sitl.launch

# Run test (in a new shell):
cd <Firmware_clone>
source integrationtests/setup_gazebo_ros.bash $(pwd)
rosrun px4 mavros_new_test.py
```

3.) 添加新的测试结点到**launch**文件

在 `launch/mavros_posix_tests_iris1.launch` 中的测试组中添加新的条目：

```
<group ns="$(arg ns)">
  [...]
  <test test-name="mavros_new_test" pkg="px4" type="mavros_new_test.py" />
</group>
```

按照前文所述方法运行完整测试套件。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

户外光流

官网英文原文地址：<http://dev.px4.io/optical-flow-outdoors.html>

本页面向您介绍如何设置PX4Flow用于位置估计以及户外自主飞行。LIDAR(激光雷达)的使用并非必要，但其的确会提升性能。

选择LPE（Local Position Estimator）估计器

唯一被测试的可以与基于户外自主飞行的光流共同作用的估计器就是LPE。

使用 `SYS_MC_EST_GROUP = 1` 参数来选择估计器然后重启飞控板。

硬件

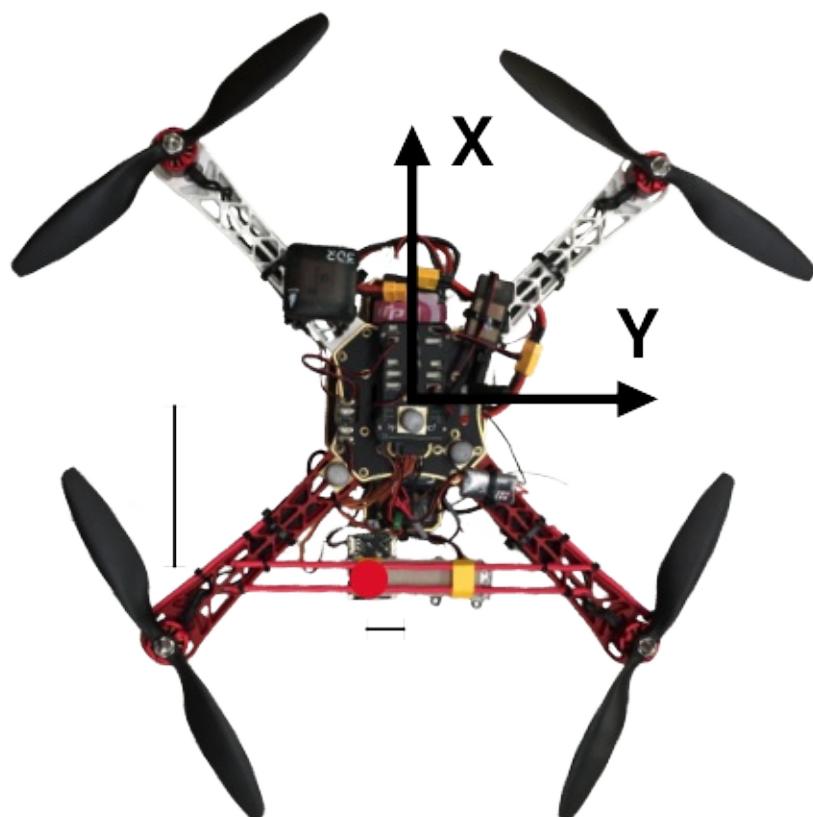


图 1: 装配坐标系（相对于下面的参数）

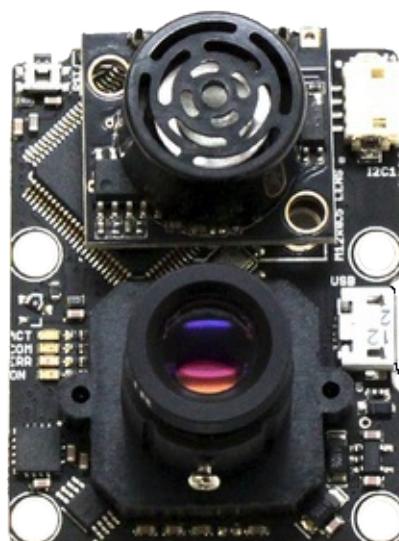


图 2: **PX4FLOW**光流传感器（相机以及声呐）

PX4Flow必须指向地面，可以使用**Pixhawk**上的I2C接口进行连接。为了使**PX4Flow**获得最好的性能，确保将其放置在一个好的位置，同时不要暴露在强烈震动环境下。（最好是将其放置在四轴飞行器的底部）

注意：**PX4Flow**放置的默认的方位是其声呐侧（光流上的+Y）指向飞行器的+X(前面)。
如果不是，则需要相应的设置 `SENS_FLOW_ROT`



图 3: **Lidar Lite**

存在包括Lidar-Lite（目前已经不生产）和 [sf10a](#) 在内的一些LIDAR设备可供选择。有关LIDAR-Lite的连接请参考[这里](#)，其中sf10a可以通过串行总线与Pixhawk相连。

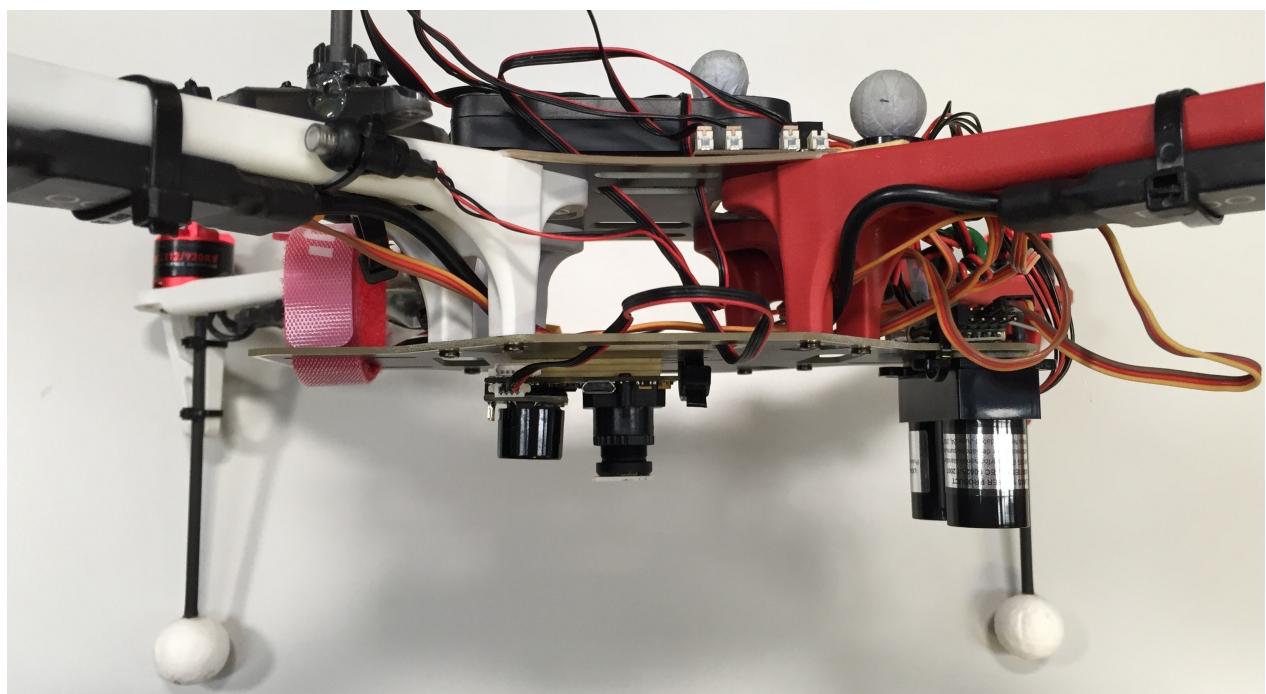


图4: 装有PX4Flow/ Lidar-Lite的DJI F450



图5: 这个Iris+上装有一个不带LIDAR的PX4Flow，其同样可以对LPE估计器起作用



图6: 为PX4Flow搭建了一个天气保护盒子。用泡沫包裹着盒子一是可以降低声呐读取的螺旋桨噪声，同时还可以保护相机免受碰撞。

相机聚焦

为了保证好的光流质量，将PX4Flow上的的相机聚焦到一个理想的飞行高度是十分重要的。要让相机聚焦，首先准备一个带有文字的物体（例如，一本书），然后将PX4Flow插入到USB中，最后运行QGroundControl。在设置菜单下，选择PX4Flow，你会看到一个相机的拍摄得到的图片。通过拧动相机的固定螺母来放松或收紧镜头找到相机的焦点的方法进行聚焦。

注意：如果你的飞行高度超过了**3米**，相机将聚焦在一个无限远的地方，对于在更高处的飞行，这一点不需要作改变



图7：用一本书在你想要飞行的高度上完成光流相机的聚焦，一般在1-3米的范围内。超过3米时，应该将相机聚焦到一个无限远的位置，这样对于在更高处的飞行也适用

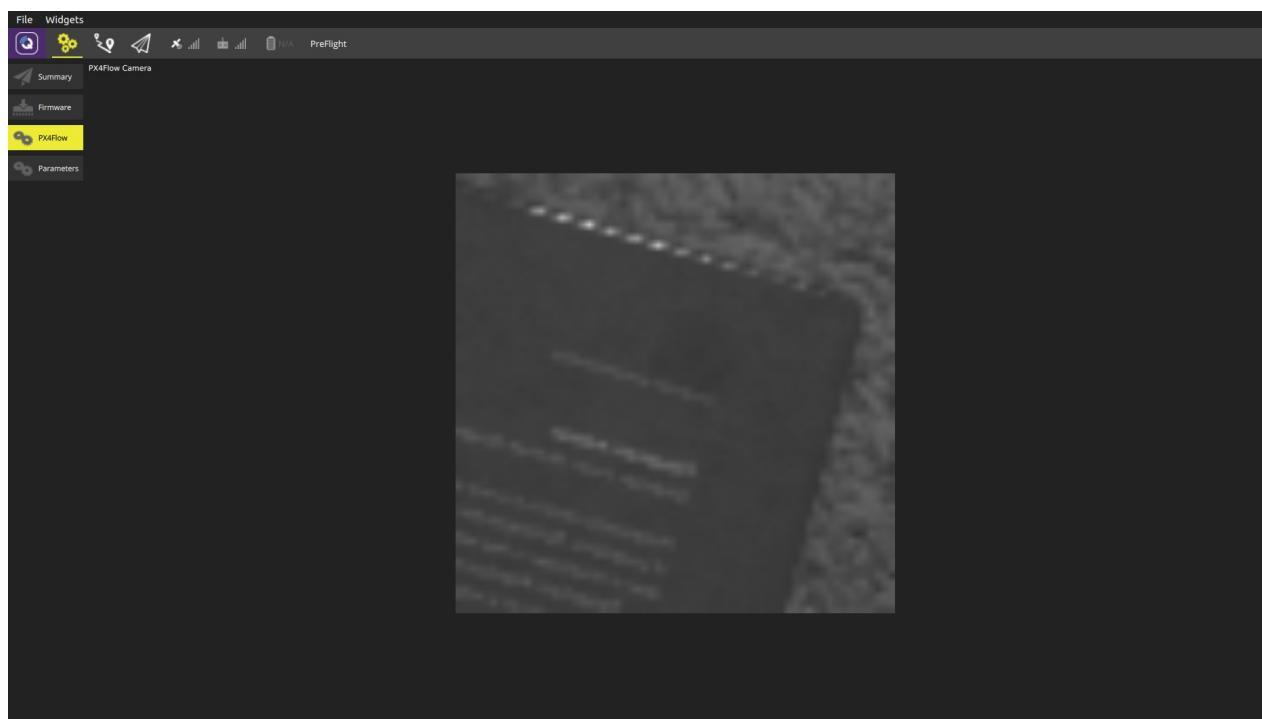


图8：QGroundControl地面站的px4flow光流界面可以被用来对相机进行聚焦

传感器参数

所有的参数都可以在QGroundControl中进行修改

- 将SEN_EN_LL40LS设置为1以使能lidar-lite进行距离测量

- 将SEN_EN_SF0X设置为1以使能lightware进行距离测量(例如.sf02和sf10a)

Local Position Estimator (LPE)

LPE是一种基于扩展卡尔曼滤波器EKF的位置与速度估计器。LPE使用了惯性导航系统并且与INAV估计器有着类似的功能，但是它能够基于状态协方差动态地计算卡尔曼增益。LPE还可以检测故障，这个功能将使类似于声呐这种能够通过软件界面返回无效测量值的传感器更好的发挥作用。

户外飞行视频

下面是一个在户外使用光流进行自主任务飞行的视频以及飞行得到的绘图。虽说没有用到GPS来对飞行器的位置进行估计，但是图中还是将GPS的信息画出来用于地面实况比较。GPS和光流数据之间的偏移是由于用户安装光流的位置的初值估计误差。初始安装位置是在LPE_LAT（经度）和LPE_LON（纬度）(在下面会进行说明)。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

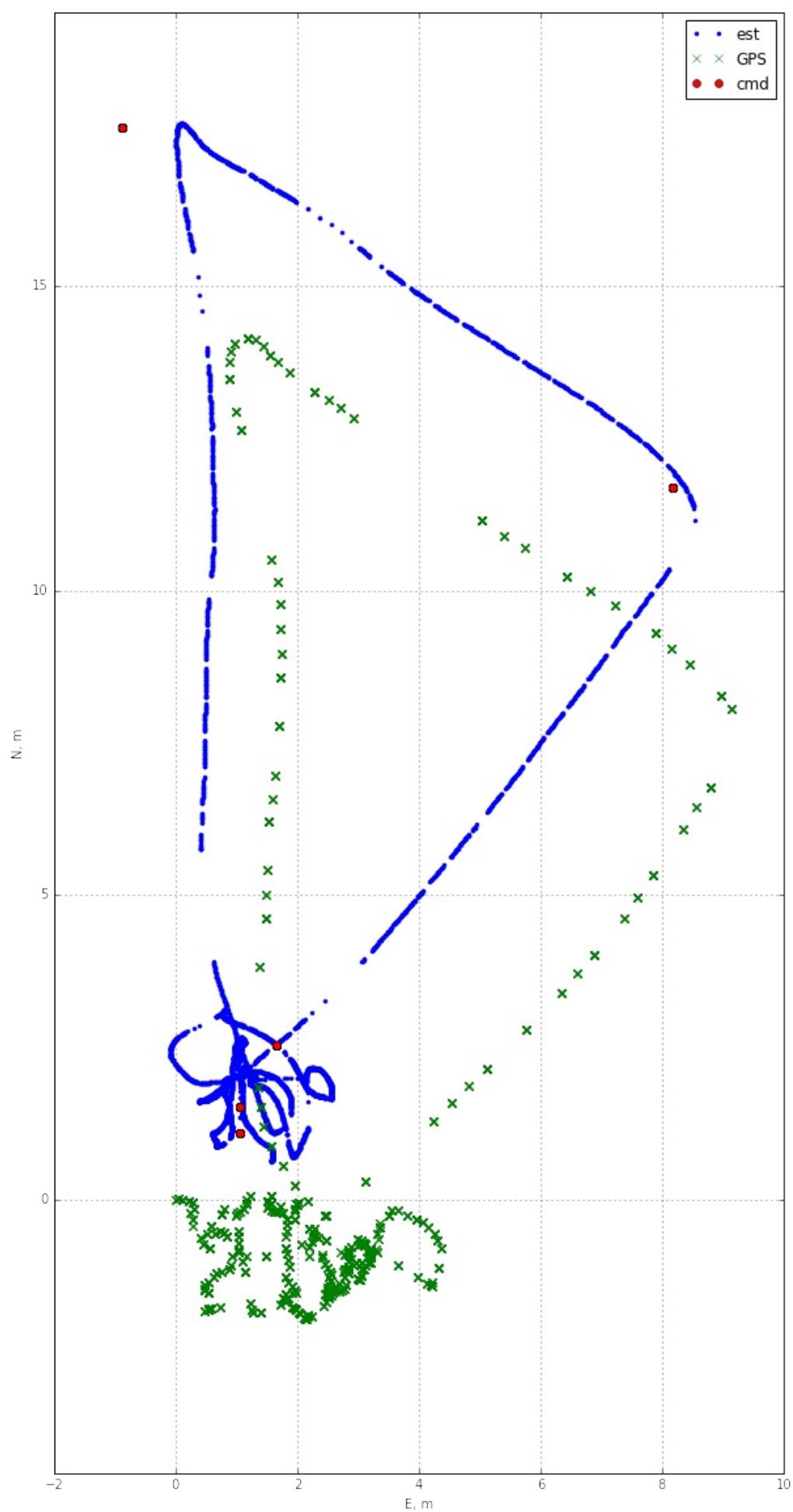


图9: 带有光流和声呐的飞行器基于LPE的自主任务飞行

参数

当传感器插入时，LPE会自动融合LIDAR和光流测量的数据。

- LPE_FLOW_OFF_Z - 这是光流相机距飞行器重心的偏移。该值向下为正，默認為0。在绝大多数正常安装的情况下，该值可以保持为0，因为Z轴上的偏移微不足道。
- LPE_FLW_XY - 光流的标准差，单位米。
- LPW_FLW_QMIN - 可接受测量的光流质量最小值。
- LPE_SNR_Z - 声呐的标准差，单位米。
- LPE_SNR_OFF_Z - 声呐传感器距飞行器重心的偏移。
- LPE_LDR_Z - 激光雷达Lidar的标准差，单位米。
- LPE_LDR_Z_OFF - Lidar距飞行器重心的偏移。
- LPE_GPS_ON - 如果参数LPE_GPS_ON设置为1，在没有GPS的情况下飞行器将无法飞行。必须禁用此项或者等待GPS获得高度信息并将位置初始化。这是为了当GPS可用时由GPS获得的高度信息优先级高于气压计获得的高度信息。

注意：在没有GPS的情况下，必须将参数LPE_GPS_ON置0才能飞行

自主飞行参数

告诉飞行器它的当前位置

- LPE_LAT - 与机体坐标系中坐标(0,0)相关联的纬度。
- LPE_LON - 与机体坐标系中坐标(0,0)相关联的经度。

让飞行器保持一个很低的高度与速度

- MPC_ALT_MODE - 将此值设置为1以使能地形跟踪
- LPE_T_Z - 这是地形的过程噪声。如果在丘陵地带飞行，将此值设置为0.1，如果是在一个平坦的停车场等类似的地方飞行，则可将此值设置为0.01。
- MPC_XY_VEL_MAX - 将此值设置为2以限制倾斜
- MPC_XY_P - 将此值降低到0.5以限制倾斜。
- MIS_TAKEOFF_ALT - 将此值设置为2米以允许低空起飞。

航点

- 在高度3米或更低处创建航点。

- 不要创建过远距离的飞行计划，每飞行100米预期会产生1米的漂移。

注意：在第一次自主飞行之前，先使用带有光流传感器的飞行器手动进行巡线，这样可以确保飞行器沿着你期望的路径飞行

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

多旋翼PID调参教程

官网英文原文地址：<http://px4.io/docs/multicopter-pid-tuning-guide/>

本教程仅适用于高级用户/专家。如果你不理解PID调参代表什么，很可能导致炸机。

绝不带着碳纤维桨或者增强型碳纤维桨进行多旋翼调参。

绝不使用损坏的桨。

鉴于安全因素，默认的增益都设置成相应的较小的值。在你想要得到控制响应前，必须先增大相应的增益。

这个教程对所有的多旋翼(AR.Drone，PWM 四轴/六轴/八轴飞行器)都有效。比例-积分-微分(PID)控制器是应用最广泛的控制技术。在模型预测控制中有一些效果更好的控制技术(LQR/LQG)，但是由于这些控制技术或多或少地需要精确的系统模型，因此没能得到广泛应用。PX4系列飞控板的控制目标就是能在MPC之间尽可能快的切换，因为不是所有支持的系统模型都可用，因此PID调参非常重要(并且PID控制在许多情况下是足够的)。

介绍

PX4中的 `multirotor_att_control` 应用程序执行外环姿态控制器的外环，取决于以下参数：

- 横滚比例控制 (MC_ROLL_P)
- 俯仰比例控制 (MC_PITCH_P)
- 偏航比例控制 (MC_YAW_P)

同时内环通过三个独立的PID控制器来控制姿态角速度：

- 横滚角速度控制(MC_ROLLRATE_P, MC_ROLLRATE_I, MC_ROLLRATE_D)
- 俯仰角速度控制(MC_PITCHRATE_P, MC_PITCHRATE_I, MC_PITCHRATE_D)
- 偏航角速度控制 (MC_YAWRATE_P, MC_YAWRATE_I, MC_YAWRATE_D)

外环输出的是期望的机体角速度(例如：如果多旋翼应该处于水平，但是当前存在30度的横滚角，那么控制输出就将产生60度每秒的角速度)。内环(即角速度控制环)改变电机的输出以使飞行器按照期望的角速度旋转。

增益实际上有一个直观的意义，例如：如果将MC_ROLL_P增益值设置为6.0并且姿态角存在0.5弧度的偏移(大约30度)，那么飞行器将尝试以6倍的角速度来进行补偿，也就是3.0弧度每秒(rad/s)或者约170度每秒(deg/s)。对于内环来说，如果将MC_ROLLRATE_P增益设置成

0.1，那么推力对横滚角的输出将会是 $3*0.1 = 0.3$ ，这意味着飞行器一侧的电机将减速30%，同时另一侧的电机加速，以此来诱导角动量以便是飞行器回到水平状态。

参数MC_YAW_FF反映的是用户输入对偏航速度控制器的前馈比例。值为0代表极慢的响应速度，仅当偏航位置误差出现时，控制器才开始偏航运动；值为1代表响应非常灵敏的控制，但有一些超调，控制器将立刻进行偏航运动并始终保持偏航误差接近零，。

电机带限 (Motor Band / Limiting)

如上述所示，在某些条件下，一个电机可能获得高于其最大速度的输入，而另一个电机获得低于零的输入。如果发生这种情况，由电机产生的力将违反控制模型，多旋翼飞行器很可能翻转。为了防止这种情况发生，PX4上的多旋翼混控器使用了一个带限。如果有一个电机超出了安全范围，系统的总推力会被降低，使得控制器输出的相对百分比能得到满足。因此，多旋翼飞行器可能不会爬升，甚至稍微掉高，但它绝不会翻转。同样对于下侧，即使命令的横滚角度很大，它也将被缩放到一个不超过命令总推力的值，并且飞行器将不会在接近零推力起飞时发生翻转。

第1步：准备

首先将所有的参数都设置为初始值：

1. 将所有的MC_XXX_P设置为0(ROLL, PITCH, YAW)
2. 除了MC_ROLLRATE_P 和 MC_PITCHRATE_P之外，将所有的MC_XXXRATE_P、MC_XXXRATE_I、MC_XXXRATE_D设置为0
3. 将MC_ROLLRATE_P和MC_PITCHRATE_P设置为一个比较小的值，例如0.02
4. 将MC_YAW_FF设置为0.5

所有的增益都应该缓慢的增加，每次增加20%~30%，在最终微调时甚至应该降至10%。注意，增益太大(即使只比最优增益值大了1.5-2倍)可能会导致非常危险的振荡！

第2步：稳定横滚(ROLL)和俯仰(PITCH)角速度

P 增益调节

参数：MC_ROLLRATE_P，MC_PITCHRATE_P

如果飞行器是对称的，那么ROLL和PITCH的值应该是相同的；如果不是——则应该分别进行调节。

用手牢牢抓住多旋翼并将油门推到大约50%，使飞机的重量几乎为零。用手让飞机在横滚或俯仰方向上倾斜，并观察其响应。飞机应该会温和的对抗该运动，但它不会试图回到水平。如果飞机发生了振荡，则需要将RATE_P调低。一旦控制响应变慢但是正确了，则继续增加

RATE_P直到飞机再次开始振荡。接下来降低**RATE_P**直到飞机只有轻微的振荡或者完全没有振荡(降低大约10%)，只剩下超调。典型值约为0.1。

D 增益调节

参数：**MC_ROLLRATE_D**, **MC_PITCHRATE_D**

假设增益处于多旋翼振荡的状态下，并且**RATE_P**略微减小。从0.01开始慢慢增加**RATE_D**，直到消除最后一点振荡。如果电机发生颤抖，那么说明**RATE_D**太大了，需要将其调低。通过调节**RATE_P**和**RATE_D**的大小，飞机的响应能够达到恰到好处。典型值大约在0.01~0.02之间。

在QGroundControl地面站中，你可以画出横滚和俯仰角速度图(**ATTITUDE.rollspeed/pitchspeed**)。这里曲线不会振荡，但是有一些超调(10%~20%)是没有关系的。

I 增益调节

如果横滚和俯仰角速度始终达不到设定值，并且存在漂移，从**MC_ROLLRATE_P**增益值的5%-10%处开始，将**MC_ROLLRATE_I**和**MC_PITCHRATE_I**增益相加

第3步： 稳定横滚和俯仰角

P增益调节

参数：**MC_RATE_P**, **MC_RATE_P**

将**MC_ROLL_P**以及**MC_PITCH_P**设置为一个较小的值，例如设置为3。

用手牢牢抓住多旋翼并将油门推到大约50%，使飞机的重量几乎为零。用手让飞机在横滚或俯仰方向上倾斜，并观察其响应。飞机应该会缓慢地回到水平。如果飞机发生振荡，则应该将P调低。一旦控制响应变慢但是正确了，则继续增加P直到飞机再次开始振荡。最优的响应是带有一些超调(大约10%~20%)。在得到稳定的响应之后再次对**RATE_P**, **RATE_D**进行微调。

在QGroundControl地面站中，你可以画出横滚和俯仰角示意图(**ATTITUDE.roll/pitch**)以及控制输出(**ctrl0, ctrl1**)。姿态角的超调量应该不超过10-20%。

第4步： 稳定偏航角速度

P增益调节

参数：**MC_YAWRATE_P**

将**MC_YAWRATE_P**设置为一个较小的值，例如设置为0.1。

用手牢牢抓住多旋翼并将油门推到大约50%，使飞机的重量几乎为零。用手让飞机在偏航方向上转动，并观察其响应。电机的声音应该会发生改变，同时飞机应该会对抗这个偏航转动。飞机的响应将基本上弱于横滚和俯仰运动，这是正常的。如果飞机发生振荡或者颤抖，则需要将RATE_P调低。如果对于很小的运动(油门到顶VS空转螺旋桨)飞机的响应也非常剧烈，则继续减少RATE_P。典型值大约在0.2~0.3之间。

如果偏航角速度控制非常灵敏或者发生振荡，可能会恶化横滚和俯仰响应。可以通过转向、横滚、俯仰和偏航检查系统的总响应。

第5步：稳定偏航角

P增益调节

参数：MC_YAW_P

将MC_YAW_P设置为一个较低的值，例如设置为1。

用手牢牢抓住多旋翼并将油门推到大约50%，使飞机的重量几乎为零。用手让飞机在偏航方向上转动，并观察其响应。飞机应该会慢慢地回到初始航向上。如果飞机发生振荡，则需要将MC_YAW_P调低。一旦控制响应变慢但是正确了，则需要增加MC_YAW_P直到响应变得稳定，同时不能有振荡。典型值大约在2~3之间。

可以在QGroundControl地面站中查看ATTITUDE.yaw偏航角曲线。偏航角的超调应该不超过2~5%。

前馈调节

参数：MC_YAW_FF

此参数不是特别重要并且可以在飞行过程中调节，在最坏的情况下，偏航响应将会滞后或者太快。调节前馈参数FF以获得一个舒适的响应。有效范围在0~1之间。典型值为0.8....0.9。(对于航拍视频，为获得平滑的响应，最佳值可能小得多)

可以在QGroundControl地面站中查看ATTITUDE.yaw偏航角曲线。偏航角的超调应该不超过2~5%。

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22
12:56:03

sdlog2

官网英文原文地址：<https://pixhawk.org//firmware//apps//sdlog2>

`sdlog2` 这个应用程序是用来将FMU的飞行数据记录到SD卡中作为日志文件。该日志文件的格式与APM的二进制日志格式兼容，但是"sdlog2" 使用强制消息"TIME" 来写时间戳。

使用

每当 `sdlog2` 开始记录时，会在 SD卡的 `log` 文件夹中创建一个新的目录。如果设置了选项 `-t` 同时有一个GPS的时间戳可用的话，文件夹的命名是基于当前的日期的(例如，`log/2014-01-19`)。否则，文件夹就会被命名为 "`sessXXX`"，这里 `xxx` 代表一个序列号。如果可能并且可以使用 `t` 选项的话，文件名的创建与使用当前时间命名的方式相似(例如，`log/2014-01-19/19_37_52.bin`)否则这个文件就命名为 `log.XXX.bin`，再次使用序列号。

根据给定的选项开始记录日志，要么当 `sdlog2` 应用程序启动，要么当系统解锁，或者通过 `mavlink`命令。

```
usage: sdlog2 {start|stop|status} [-r <log rate>] [-b <buffer size>] -e -a
      -r      Log rate in Hz, 0 means unlimited rate
      -b      Log buffer size in KBytes, default is 8
      -e      Enable logging on app start (if not, can be started by command)
      -a      Log only when armed (can be still overridden by command)
      -t      Use GPS timestamps to create folder and file names
```

该日志记录的性能取决于所使用的microSD卡。建议使用高质量的SD卡以避免遗漏数据。虽说不会对飞行性能产生负面影响，但是全速运行"sdlog2"应用程序（即不加 `-1` 选项）可能会引起巨大的CPU负载。然而，所需的全速可能无法得到满足，日志记录也将跳过一些消息。

开始记录日志

在一般情况下，飞行器解锁之后就会开始记录日志，因为只有激活了的飞行才值得分析。如果要手动启动日志记录，可以在**console**控制台输入以下指令：

```
sdlog2 stop

sdlog2 start -t -r 200 -e -b 16
```

要停止记录则输入：

```
sdlog2 stop
```

分析日志文件

FlightPlot

要查看以及分析日志，可以使用GUI工具 [FlightPlot](#)。它可以无需转换地读取 `sdlog2` 生成的日志文件.。

Pymavlink

你也可以使用包含在 [PyMAVLink](#) 中的mavgraph工具来生成绘图。

CSV / Matlab: Converting Logs to CSV

要读取二进制文件并将其转换为CSV，可以使用Python工具[sdlog2_dump.py](#)。同样的目录中包含着运行转换器和绘制大量核心信息的MATLAB脚本。

例如：要读取 `TIME` 和 `IMU` 的消息，`SENS` 消息中的 `BaroAlt` 和 `BaroTemp` 数据，请使用下面的指令：

```
python sdlog2_dump.py log001.px4log -t TIME -m TIME -m IMU -m SENS.BaroAlt,BaroTemp
```

要创建CSV直接重定向输出到文件：

```
python sdlog2_dump.py log001.px4log -t TIME -m TIME -m IMU -m SENS.BaroAlt,BaroTemp > log
```

CSV文件中的列将与参数具有相同的顺序。选项 `t` 显著的减少了输出的重复数据，应该始终被 `sdlog2` 生成的日志记录文件使用。但是不要用在原来的APM日志文件中。

消息类型举例

- TIME - 时间戳
- ATT - 飞行器的姿态
- ATSP - 飞行器的姿态设定值
- IMU - IMU传感器
- SENS - 其他传感器
- LPOS - 本地位置估计
- LPSP - 本地位置设定值
- GPS - GPS位置
- ATTC - 姿态控制 (actuator_0 output)
- STAT - 飞行器的状态
- RC - 遥控输入通道
- OUT0 - Actuator_0 output
- AIRS - 空速
- ARSP - 角速度设定值
- FLOW - 光流
- GPOS - 全球位置估计
- GPSP - 全球位置设定值
- ESC - 电调状态
- GVSP - 全球速度设定值

消息类型有时是可以改变的。为了找出包含在日志文件中的实际消息列表，可以使用以下的指令：

```
python sdlog2_dump.py log001.bin -v
```

或者使用[FlightPlot](#)来查看。

发现并修理故障

为了防止在飞行过程中有大量的IO操作时关键的应用程序终止，`sdlog2` 在接收消息和将日志写到SD卡之间设置了一个缓冲区。如果缓冲器在一些点溢出了，一些日志消息将被跳过。跳过的消息数可以通过在控制台中使用`sdlog2 status` 指令进行检查。以及关闭日志文件时打印的统计数字，即，在解锁时使用的`-a` 选项。如果跳过的消息数不是0，就可以通过在`-b` 选项中增加缓冲器大小进行修复。以下指令将设置日志缓冲区的大小为16KiB,而不是默认的8KiB:

```
sdlog2 start -t -r 100 -e -b 16
```

负载测试

为了测试microSD的带宽，以200Hz的频率32KB的缓冲区开启该应用程序，键入 `-e` 标志立刻开始记录日志。

```
#首先停止正在运行的实例  
sdlog2 stop  
sdlog2 start -t -r 200 -e -b 32  
# 运行perf命令以查看sdlog2产生的负载  
# (注意：性能计数器仅在日志记录期间存在)  
perf  
# 或者直接运行top(在Ubuntu用用来查看当前系统资源)  
top  
# 停止应用程序以清理文件描述符和文件系统  
sdlog2 stop
```

日志文件样本

- [downloads `sdlog2_sample_log_001.bin.zip`](#) - captured at 100 Hz rate

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22 12:56:03

使用 ecl EKF

官网英文原文地址：http://dev.px4.io/tuning_the_ecl_ekf.html

本教程旨在解答一些关于ECL EKF算法使用的常见问题。

什么是 ecl EKF?

ECL(Estimation and Control Library，估计与控制库) 使用EKF(Extended Kalman Filter，扩展卡尔曼滤波器)算法来处理传感器测量信息并为下面的状态提供估计：

- 四元数，定义为从地球NED(北东地)坐标系到机体坐标系X,Y,Z的旋转四元数
- 速度，关于IMU - 北，东，地 (m/s)
- 位置，关于IMU - 北，东，地 (m)
- 角速度偏移估计，关于IMU - X,Y,Z (rad)
- 速度偏移估计，关于IMU - X,Y,Z (m/s)
- 地磁分量 - 北，东，地 (gauss)
- 磁偏量，关于飞行器本身 - X,Y,Z (gauss)
- 风速 - 北，东 (m/s)

EKF在一个延迟的 `fusion time horizon` 上运行，以允许传感器在每次测量时相对于IMU存在不同的时间延迟。每个传感器的数据都是FIFO缓存的并由EKF从缓存中检索，得以在正确的时候使用。每个传感器的延迟补偿由参数 `EKF2_*_DELAY` 控制。

互补滤波器用于利用缓存好的IMU数据将状态从 `fusion time horizon` 向前传播到当前时间。该滤波器的时间常数由参数 `EKF2_TAU_VEL` 和 `EKF2_TAU_POS` 控制。

注意： `fusion time horizon` 延迟和缓冲区的长度由参数 `EKF2_*_DELAY` 的最大值确定。如果未使用某传感器，建议将其时间延迟设置为零。降低 `fusion time horizon` 延迟会减少互补滤波器中用于将状态向前传播到当前时间的误差

调整位置和速度状态以消除IMU和机体坐标系之间由于安装误差所产生的偏移，避免其输出到控制回路。IMU相对于机体坐标系的位置由参数 `EKF2_IMU_POS_X, Y, Z` 设置。

EKF仅使用IMU的数据进行状态预测。IMU的数据不会用作EKF推导过程中的量测值。协方差预测、状态更新以及协方差更新的代数方程都是使用Matlab符号工具箱导出的，可以在这里找到：[Matlab Symbolic Derivation](#)

ecl EKF使用何种传感器测量值？

根据传感器测量值的不同组合，EKF具有不同的操作模式。在启动时，滤波器会检查传感器的最小可行组合，并在初始倾斜、偏航以及高度对准完成后进入一个提供旋转、垂直速度、垂直位置、IMU角增量误差和IMU速度增量误差估计的模式。

此模式需要有传感器的数据，例如偏航数据源（由磁力计或者外部视觉设备提供）和高度数据源。所有的EKF操作模式都需要这个最小的数据集。然后可以使用其他传感器数据来估计附加的状态。

惯性测量单元 (IMU)

- 惯性测量单元的三轴位置固定，用于采集单位角增量和速度增量数据，最低采样频率为100Hz。

注意：在EKF使用IMU角增量数据之前应对其进行校正修正。

磁力计

三轴磁力计（或者是外部视觉系统）的最低采样率为5Hz。磁力计数据可以以两种方式使用：

- Magnetometer measurements are converted to a yaw angle using the tilt estimate and magnetic declination. This yaw angle is then used as an observation by the EKF. This method is less accurate and does not allow for learning of body frame field offsets, however it is more robust to magnetic anomalies and large start-up gyro biases. It is the default method used during start-up and on ground.
- 使用倾斜估计和磁偏角将磁力计测量值转换为偏航角
- The XYZ magnetometer readings are used as separate observations. This method is more accurate and allows body frame offsets to be learned, but assumes the earth magnetic field environment only changes slowly and performs less well when there are significant external magnetic anomalies. It is the default method when the vehicle is airborne and has climbed past 1.5 m altitude.

The logic used to select the mode is set by the EKF2_MAG_TYPE parameter.

Height

A source of height data - either GPS, barometric pressure, range finder or external vision at a minimum rate of 5Hz is required. Note: The primary source of height data is controlled by the EKF2_HGT_MODE parameter.

If these measurements are not present, the EKF will not start. When these measurements have been detected, the EKF will initialise the states and complete the tilt and yaw alignment. When tilt and yaw alignment is complete, the EKF can then transition to other modes of operation enabling use of additional sensor data:

GPS

GPS measurements will be used for position and velocity if the following conditions are met:

- GPS use is enabled via setting of the EKF2_AID_MASK parameter.
- GPS quality checks have passed. These checks are controlled by the EKF2_GPS_CHECK and EKF2_REQ<> parameters.
- GPS height can be used directly by the EKF via setting of the EKF2_HGT_MODE parameter.

Range Finder

Range finder distance to ground is used a by a single state filter to estimate the vertical position of the terrain relative to the height datum.

If operating over a flat surface that can be used as a zero height datum, the range finder data can also be used directly by the EKF to estimate height by setting the EKF2_HGT_MODE parameter to 2.

Airspeed

Equivalent Airspeed (EAS) data can be used to estimate wind velocity and reduce drift when GPS is lost by setting EKF2_ARSP_THR to a positive value. Airspeed data will be used when it exceeds the threshold set by a positive value for EKF2_ARSP_THR and the vehicle type is not rotary wing.

Synthetic Sideslip

Fixed wing platforms can take advantage of an assumed sideslip observation of zero to improve wind speed estimation and also enable wind speed estimation without an airspeed sensor. This is enabled by setting the EKF2_FUSE_BETA parameter to 1.

Optical Flow

Optical flow data will be used if the following conditions are met:

- Valid range finder data is available.
- Bit position 1 in the EKF2_AID_MASK parameter is true.
- The quality metric returned by the flow sensor is greater than the minimum requirement set by the EKF2_OF_QMIN parameter

External Vision System

Position and Pose Measurements from an external vision system, eg Vicon, can be used:

- External vision system horizontal position data will be used if bit position 3 in the EKF2_AID_MASK parameter is true.
 - External vision system vertical position data will be used if the EKF2_HGT_MODE parameter is set to 3.
- * External vision system pose data will be used for yaw estimation if bit position 4 in the EKF2_AID_MASK parameter is true.

How do I use the 'ecl' library EKF?

Set the SYS_MC_EST_GROUP parameter to 2 to use the ecl EKF.

What are the advantages and disadvantages of the ecl EKF over other estimators?

Like all estimators, much of the performance comes from the tuning to match sensor characteristics. Tuning is a compromise between accuracy and robustness and although we have attempted to provide a tune that meets the needs of most users, there will be applications where tuning changes are required.

For this reason, no claims for accuracy relative to the legacy combination of attitude_estimator_q + local_position_estimator have been made and the best choice of estimator will depend on the application and tuning.

Disadvantages

* The ecl EKF is a complex algorithm that requires a good understanding of extended Kalman filter theory and its application to navigation problems to tune successfully. It is therefore more difficult for users that are not achieving good results to know what to change.

- The ecl EKF uses more RAM and flash space
- The ecl EKF uses more logging space.
- The ecl EKF has had less flight time

Advantages

- The ecl EKF is able to fuse data from sensors with different time delays and data rates in a mathematically consistent way which improves accuracy during dynamic

manoeuvres once time delay parameters are set correctly.

- The ecl EKF is capable of fusing a large range of different sensor types.
- The ecl EKF detects and reports statistically significant inconsistencies in sensor data, assisting with diagnosis of sensor errors.
- For fixed wing operation, the ecl EKF estimates wind speed with or without an airspeed sensor and is able to use the estimated wind in combination with airspeed measurements and sideslip assumptions to extend the dead-reckoning time available if GPS is lost in flight.
- The ecl EKF estimates 3-axis accelerometer bias which improves accuracy for tailsitters and other vehicles that experience large attitude changes between flight phases.
- The federated architecture (combined attitude and position/velocity estimation) means that attitude estimation benefits from all sensor measurements. This should provide the potential for improved attitude estimation if tuned correctly.

How do I check the EKF performance?

EKF outputs, states and status data are published to a number of uORB topics which are logged to the SD card during flight. The following guide assumes that data has been logged using the .ulog file format. To use the .ulog format, set the SYS_LOGGER parameter to 1.

The .ulog format data can be parsed in python by using the [PX4 pyulog library](#).

Most of the EKF data is found in the `ekf2_innovations` and `estimator_status` uORB messages that are logged to the .ulog file.

Output Data

- Attitude output data is found in the `vehicle_attitude` message.
- Local position output data is found in the `vehicle_local_position` message.
- Control loop feedback data is found in the `control_state` message.
- Global (WGS-84) output data is found in the `vehicle_global_position` message.
- Wind velocity output data is found in the `wind_estimate` message.

States

Refer to states[32] in `estimator_status`. The index map for states[32] is as follows:

- [0 ... 3] Quaternions
- [4 ... 6] Velocity NED (m/s)
- [7 ... 9] Position NED (m)
- [10 ... 12] IMU delta angle bias XYZ (rad)

- [13 ... 15] IMU delta velocity bias XYZ (m/s)
- [16 ... 18] Earth magnetic field NED (gauss)
- [19 ... 21] Body magnetic field XYZ (gauss)
- [22 ... 23] Wind velocity NE (m/s)
- [24 ... 32] Not Used

State Variances

Refer to covariances[28] in [estimator_status](#). The index map for covariances[28] is as follows:

- [0 ... 3] Quaternions
- [4 ... 6] Velocity NED (m/s)²
- [7 ... 9] Position NED (m²)
- [10 ... 12] IMU delta angle bias XYZ (rad²)
- [13 ... 15] IMU delta velocity bias XYZ (m/s)²
- [16 ... 18] Earth magnetic field NED (gauss²)
- [19 ... 21] Body magnetic field XYZ (gauss²)
- [22 ... 23] Wind velocity NE (m/s)²
- [24 ... 28] Not Used

Observation Innovations

- Magnetometer XYZ (gauss) : Refer to [mag_innov\[3\]](#) in [ekf2_innovations](#).
- Yaw angle (rad) : Refer to [heading_innov](#) in [ekf2_innovations](#).
- Velocity and position innovations : Refer to [vel_pos_innov\[6\]](#) in [ekf2_innovations](#). The index map for [vel_pos_innov\[6\]](#) is as follows:
 - [0 ... 2] Velocity NED (m/s)
 - [3 ... 5] Position NED (m)
 - True Airspeed (m/s) : Refer to [airspeed_innov](#) in [ekf2_innovations](#).
 - Synthetic sideslip (rad) : Refer to [beta_innov](#) in [ekf2_innovations](#).
 - Optical flow XY (rad/sec) : Refer to [flow_innov](#) in [ekf2_innovations](#).
 - Height above ground (m) : Refer to [hagl_innov](#) in [ekf2_innovations](#).

Observation Innovation Variances

- Magnetometer XYZ (gauss²) : Refer to [mag_innov_var\[3\]](#) in [ekf2_innovations](#).
- Yaw angle (rad²) : Refer to [heading_innov_var](#) in the [ekf2_innovations](#) message.
- Velocity and position innovations : Refer to [vel_pos_innov_var\[6\]](#) in [ekf2_innovations](#). The index map for [vel_pos_innov_var\[6\]](#) is as follows:
 - [0 ... 2] Velocity NED (m/s)²

- [3 ... 5] Position NED (m^2)
- True Airspeed ($m/s)^2$: Refer to `airspeed_innov_var` in `ekf2_innovations`.
- Synthetic sideslip (rad^2) : Refer to `beta_innov_var` in `ekf2_innovations`.
- Optical flow XY ($rad/sec)^2$: Refer to `flow_innov_var` in `ekf2_innovations`.
- Height above ground (m^2) : Refer to `hagl_innov_var` in `ekf2_innovations`.

Output Complementary Filter

The output complementary filter is used to propagate states forward from the fusion time horizon to current time. To check the magnitude of the angular, velocity and position tracking errors measured at the fusion time horizon, refer to `output_tracking_error[3]` in the `ekf2_innovations` message. The index map is as follows:

- [0] Angular tracking error magnitude (rad)
- [1] Velocity tracking error magnitude (m/s). The velocity tracking time constant can be adjusted using the `EKF2_TAU_VEL` parameter. Reducing this parameter reduces steady state errors but increases the amount of observation noise on the NED velocity outputs.
- [2] Position tracking error magnitude (m). The position tracking time constant can be adjusted using the `EKF2_TAU_POS` parameter. Reducing this parameter reduces steady state errors but increases the amount of observation noise on the NED position outputs.

EKF Errors

The EKF contains internal error checking for badly conditioned state and covariance updates. Refer to the `filter_fault_flags` in `estimator_status`.

Observation Errors

There are two categories of observation faults:

- Loss of data. An example of this is a range finder failing to provide a return.
- The innovation, which is the difference between the state prediction and sensor observation is excessive. An example of this is excessive vibration causing a large vertical position error, resulting in the barometer height measurement being rejected.

Both of these can result in observation data being rejected for long enough to cause the EKF to attempt a reset of the states using the sensor observations. All observations have a statistical confidence check applied to the innovations. The number of standard deviations for the check are controlled by the `EKF2_<>_GATE` parameter for each observation type.

Test levels are available in [estimator_status](#) as follows:

- mag_test_ratio : ratio of the largest magnetometer innovation component to the innovation test limit
- vel_test_ratio : ratio of the largest velocity innovation component to the innovation test limit
- pos_test_ratio : ratio of the largest horizontal position innovation component to the innovation test limit
- hgt_test_ratio : ratio of the vertical position innovation to the innovation test limit
- tas_test_ratio : ratio of the true airspeed innovation to the innovation test limit
- hagl_test_ratio : ratio of the height above ground innovation to the innovation test limit

For a binary pass/fail summary for each sensor, refer to innovation_check_flags in [estimator_status](#).

GPS Quality Checks

The EKF applies a number of GPS quality checks before commencing GPS aiding. These checks are controlled by the EKF2_GPS_CHECK and EKF2_REQ<> parameters. The pass/fail status for these checks is logged in the [estimator_status.gps_check_fail_flags](#) message. This integer will be zero when all required GPS checks have passed. If the EKF is not commencing GPS alignment, check the value of the integer against the bitmask definition [gps\check_fail_flags](#) in [estimator_status](#).

EKF Numerical Errors

The EKF uses single precision floating point operations for all of its computations and first order approximations for derivation of the covariance prediction and update equations in order to reduce processing requirements. This means that it is possible when re-tuning the EKF to encounter conditions where the covariance matrix operations become badly conditioned enough to cause divergence or significant errors in the state estimates.

To prevent this, every covariance and state update step contains the following error detection and correction steps:

- If the innovation variance is less than the observation variance (this requires a negative state variance which is impossible) or the covariance update will produce a negative variance for any of the states, then:
 - The state and covariance update is skipped
 - The corresponding rows and columns in the covariance matrix are reset
 - The failure is recorded in the [estimator_status filter_fault_flags](#) message
- State variances (diagonals in the covariance matrix) are constrained to be non-negative.

- An upper limit is applied to state variances.
- Symmetry is forced on the covariance matrix.

After re-tuning the filter, particularly re-tuning that involve reducing the noise variables, the value of `estimator_status.gps_check_fail_flags` should be checked to ensure that it remains zero.

What should I do if the height estimate is diverging?

The most common cause of EKF height diverging away from GPS and altimeter measurements during flight is clipping and/or aliasing of the IMU measurements caused by vibration. If this is occurring, then the following signs should be evident in the data

- `ekf2_innovations.vel_pos_innov[3]` and `ekf2_innovations.vel_pos_innov[5]` will both have the same sign.
- `estimator_status.hgt_test_ratio` will be greater than 1.0

The recommended first step is to ensure that the autopilot is isolated from the airframe using an effective isolation mounting system. An isolator mount has 6 degrees of freedom, and therefore 6 resonant frequencies. As a general rule, the 6 resonant frequencies of the autopilot on the isolation mount should be above 25Hz to avoid interaction with the autopilot dynamics and below the frequency of the motors.

An isolation mount can make vibration worse if the resonant frequencies coincide with motor or propeller blade passage frequencies.

The EKF can be made more resistant to vibration induced height divergence by making the following parameter changes:

- Double the value of the innovation gate for the primary height sensor. If using barometric height this is `EKF2_BARO_GATE`.
- Increase the value of `EKF2_ACC_NOISE` to 0.5 initially. If divergence is still occurring, increase in further increments of 0.1 but do not go above 1.0

Note that the effect of these changes will make the EKF more sensitive to errors in GPS vertical velocity and barometric pressure.

What should I do if the position estimate is diverging?

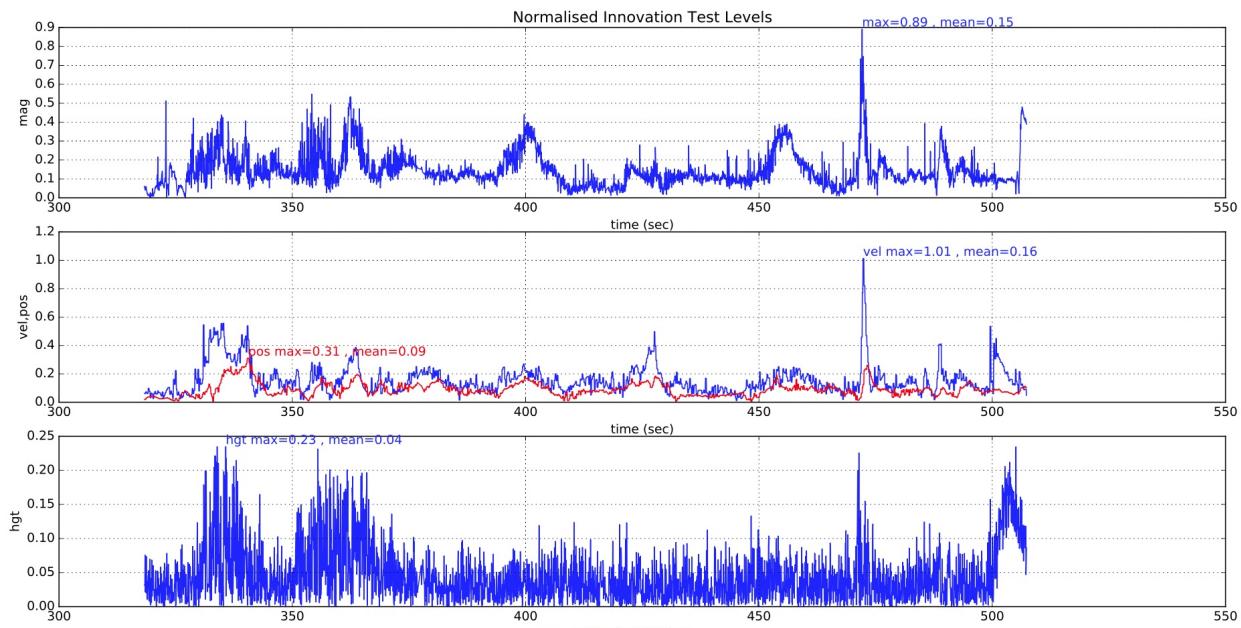
The most common causes of position divergence are:

- High vibration levels.
 - Fix by improving mechanical isolation of the autopilot.
 - Increasing the value of EKF2_ACC_NOISE and EKF2_GYR_NOISE can help, but does make the EKF more vulnerable to GPS glitches.
- Large gyro bias offsets.
 - Fix by re-calibrating the gyro. Check for excessive temperature sensitivity (> 3 deg/sec bias change during warm-up from a cold start and replace the sensor if affected or insulate to slow the rate of temeprature change.
- Bad yaw alignment
 - Check the magntometer calibration and alignment.
 - Check the heading shown QGC is within within 15 deg truth
- Poor GPS accuracy
 - Check for interference
 - Improve separation and shielding
 - Check flying location for GPS signal obstructions and reflectors (nearboy tall buildings)
- Loss of GPS

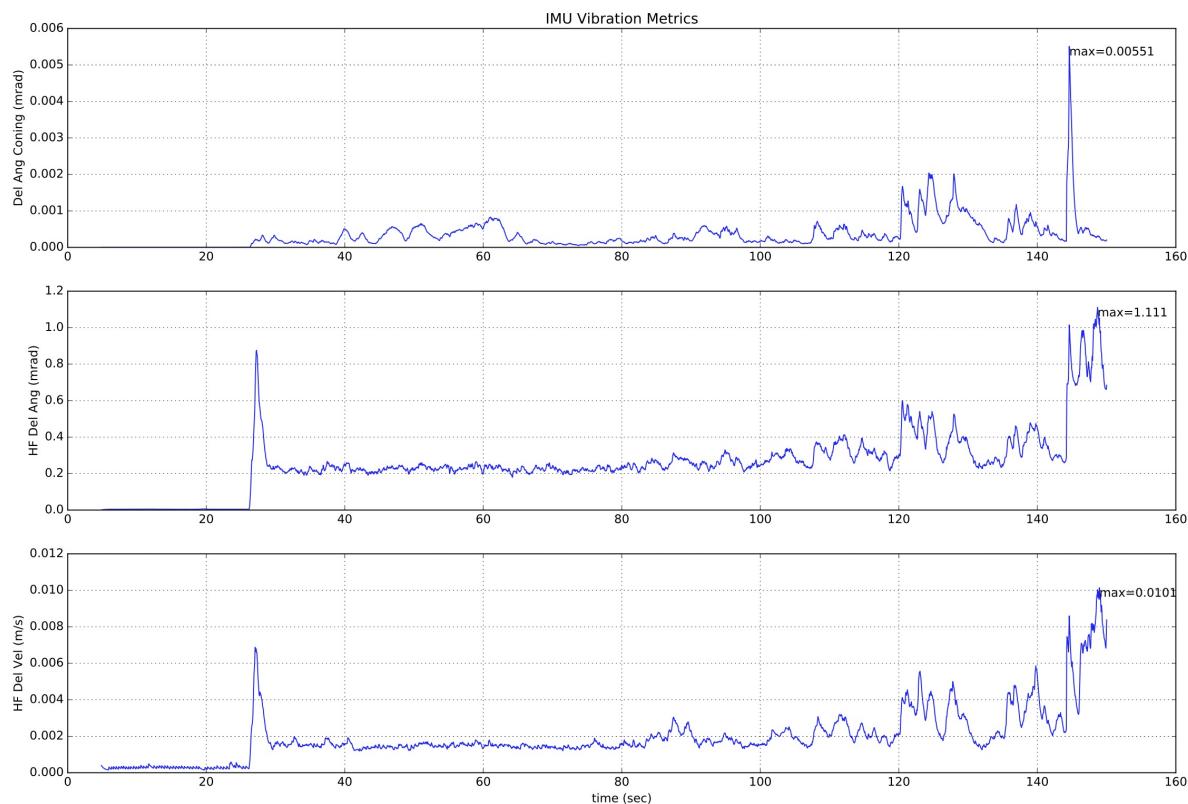
Determining which of these is the primary casue requires a methodical approach to analysis of the EKF log data:

- Plot the velocity innovation test ratio - `estimator_status.vel_test_ratio`)
- Plot the horizontal position innovation test ratio - `estimator_status.pos_test_ratio`)
- Plot the height innovation test ratio - `estimator_status.hgt_test_ratio`)
- Plot the magnetoemrer innovation test ratio - `estimator_status.mag_test_ratio`)
- Plot the GPS receier reported speed accuracy - `vehicle_gps_position.s_variance_m_s`)
- Plot the IMU delta angle state estimates - `estimator_status.states[10],states[11]` and `states[12]`
- Plot the EKF internal high frequency vibration metrics:
 - Delta angle coning vibration -`estimator_status.vibe[0]`
 - High frequency delta angle vibration - `estimator_status.vibe[1]`
 - High frequency delta velocity vibration - `estimator_status.vibe[2]`

During normal operation, all the test ratios should remain below 0.5 with only occasional spikes above this as shown in the example below from a successful flight:



The following plot shows the EKF vibration metrics for a multirotor with good isolation. The landing shock and the increased vibration during takeoff and landing can be seen. Insufficient data has been gathered with these metrics to provide specific advice on maximum thresholds.



The above vibration metrics are of limited value as the presence of vibration at a frequency close to the IMU sampling frequency (1kHz for most boards) will cause offsets to appear in the data that do not show up in the high frequency vibration metrics. The only way to detect aliasing errors is in their effect on inertial navigation accuracy and the rise in innovation levels.

In addition to generating large position and velocity test ratios of > 1.0, the different error mechanisms affect the other test ratios in different ways:

Determination of Excessive Vibration

High vibration levels normally affect vertical position and velocity innovations as well as the horizontal components. Magnetometer test levels are only affected to a small extent.

(insert example plots showing bad vibration here)

Determination of Excessive Gyro Bias

Large gyro bias offsets are normally characterised by a change in the value of delta angle bias greater than 5E-4 during flight (equivalent to ~3 deg/sec) and can also cause a large increase in the magnetometer test ratio if the yaw axis is affected. Height is normally unaffected other than extreme cases. Switch on bias value of up to 5 deg/sec can be tolerated provided the filter is given time to settle before flying. Pre-flight checks performed by the commander should prevent arming if the position is diverging.

(insert example plots showing bad gyro bias here)

Determination of Poor Yaw Accuracy

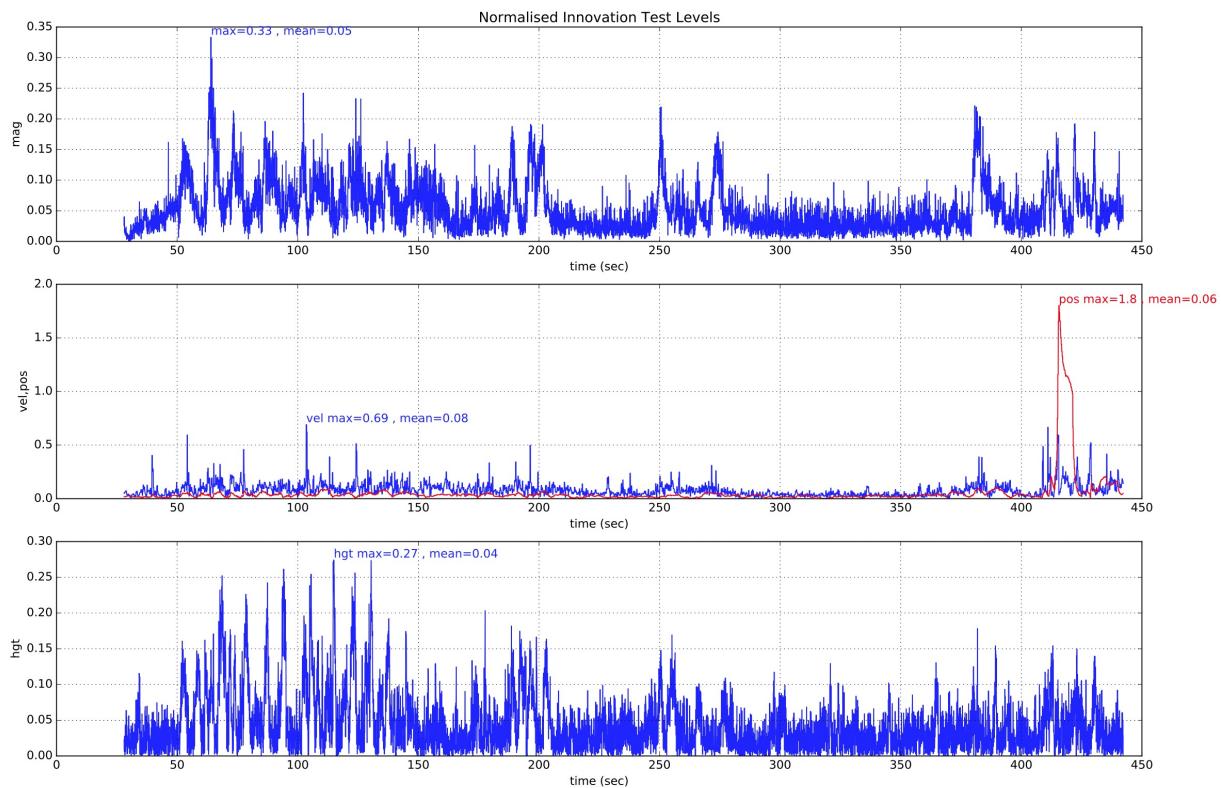
Bad yaw alignment causes a velocity test ratio that increases rapidly when the vehicle starts moving due to inconsistency in the direction of velocity calculated by the inertial nav and the GPS measurement. Magnetometer innovations are slightly affected. Height is normally unaffected.

(insert example plots showing bad yaw alignment here)

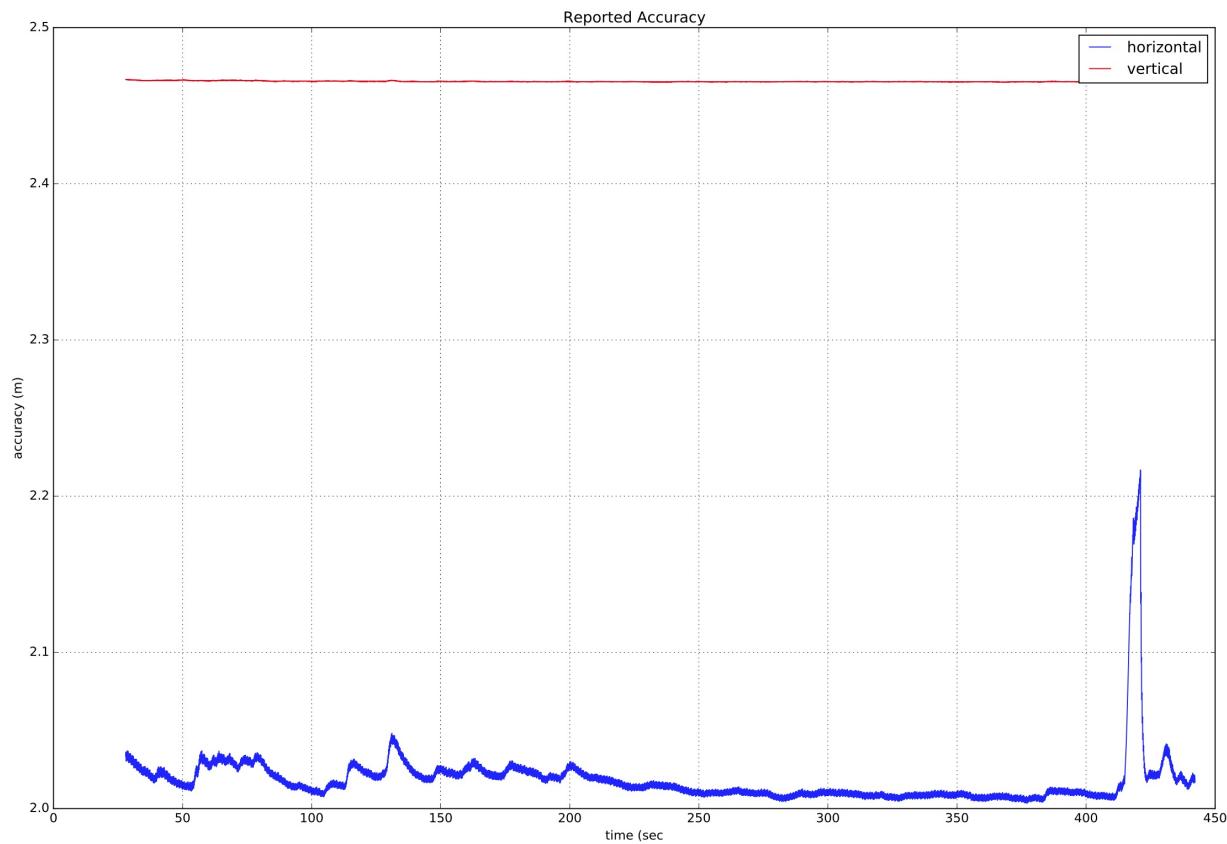
Determination of Poor GPS Accuracy

Poor GPS accuracy is normally accompanied by a rise in the reported velocity error of the receiver in conjunction with a rise in innovations. Transient errors due to multipath, obscuration and interference are more common causes. Here is an example of a temporary loss of GPS accuracy where the multi-rotor started drifting away from its loiter location and

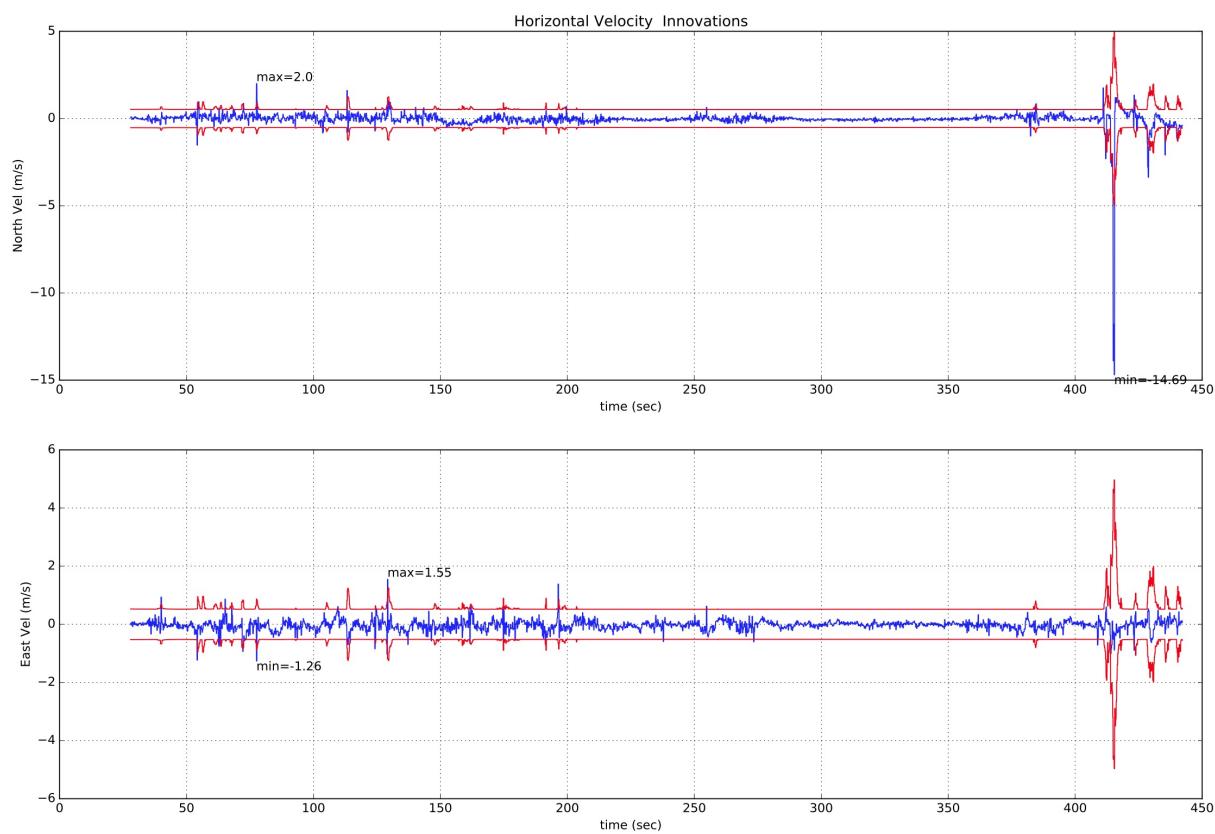
had to be corrected using the sticks. The rise in `estimator_status.vel_test_ratio`) to greater than 1 indicates the GPs velocity was inconsistent with other measurements and has been rejected.



This is accompanied with rise in the GPS receivers reported velocity accuracy which indicates that it was likely a GPS error.



If we also look at the GPS horizontal velocity innovations and innovation variances, we can see the large spike in North velocity innovation that accompanies this GPS 'glitch' event.



Determination of GPS Data Loss

Loss of GPS data will be shown by the velocity and position innovation test ratios 'flat-lining'. If this occurs, check the other GPS status data in vehicle_gps_position for further information.

(insert example plots showing loss of GPS data here)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Preflight Sensor and EKF Checks

官网英文原文地址：http://dev.px4.io/pre_flight_checks.html

The commander module performs a number of preflight sensor quality and EKF checks which are controlled by the COM_ARM<> parameters. If these checks fail, the motors are prevented from arming and the following error messages are produced:

- PREFLIGHT FAIL: EKF HGT ERROR
 - This error is produced when the IMU and height measurement data are inconsistent.
 - Perform an accel and gyro calibration and restart the vehicle. If the error persists, check the height sensor data for problems.
 - The check is controlled by the COM_ARM_EKF_HGT parameter.
- PREFLIGHT FAIL: EKF VEL ERROR
 - This error is produced when the IMU and GPS velocity measurement data are inconsistent.
 - Check the GPS velocity data for un-realistic data jumps. If GPS quality looks OK, perform an accel and gyro calibration and restart the vehicle.
 - The check is controlled by the COM_ARM_EKF_VEL parameter.
- PREFLIGHT FAIL: EKF HORIZ POS ERROR
 - This error is produced when the IMU and position measurement data (either GPS or external vision) are inconsistent.
 - Check the position sensor data for un-realistic data jumps. If data quality looks OK, perform an accel and gyro calibration and restart the vehicle.
 - The check is controlled by the COM_ARM_EKF_POS parameter.
- PREFLIGHT FAIL: EKF YAW ERROR
 - This error is produced when the yaw angle estimated using gyro data and the yaw angle from the magnetometer or external vision system are inconsistent.
 - Check the IMU data for large yaw rate offsets and check the magnetometer alignment and calibration.
 - The check is controlled by the COM_ARM_EKF_POS parameter
- PREFLIGHT FAIL: EKF HIGH IMU ACCEL BIAS
 - This error is produced when the IMU accelerometer bias estimated by the EKF is excessive.
 - The check is controlled by the COM_ARM_EKF_AB parameter.

- PREFLIGHT FAIL: EKF HIGH IMU GYRO BIAS
 - This error is produced when the IMU gyro bias estimated by the EKF is excessive.
 - The check is controlled by the COM_ARM_EKF_GB parameter.
- PREFLIGHT FAIL: ACCEL SENSORS INCONSISTENT - CHECK CALIBRATION
 - This error message is produced when the acceleration measurements from different IMU units are inconsistent.
 - This check only applies to boards with more than one IMU.
 - The check is controlled by the COM_ARM_IMU_ACC parameter.
- PREFLIGHT FAIL: GYRO SENSORS INCONSISTENT - CHECK CALIBRATION
 - This error message is produced when the angular rate measurements from different IMU units are inconsistent.
 - This check only applies to boards with more than one IMU.
 - The check is controlled by the COM_ARM_IMU_GYR parameter.

COM_ARM_WO_GPS

The COM_ARM_WO_GPS parameter controls whether arming is allowed without a GPS signal. This parameter must be set to 0 to allow arming when there is no GPS signal present. Arming without GPS is only allowed if the flight mode selected does not require GPS.

COM_ARM_EKF_POS

The COM_ARM_EKF_POS parameter controls the maximum allowed inconsistency between the EKF inertial measurements and position reference (GPS or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_VEL

The COM_ARM_EKF_VEL parameter controls the maximum allowed inconsistency between the EKF inertial measurements and GPS velocity measurements. The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_HGT

The COM_ARM_EKF_HGT parameter controls the maximum allowed inconsistency between the EKF inertial measurements and height measurement (Baro, GPS, range finder or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_YAW

The COM_ARM_EKF_YAW parameter controls the maximum allowed inconsistency between the EKF inertial measurements and yaw measurement (magnetometer or external vision). The default value of 0.5 allows the differences to be no more than 50% of the maximum tolerated by the EKF and provides some margin for error increase when flight commences.

COM_ARM_EKF_AB

The COM_ARM_EKF_AB parameter controls the maximum allowed EKF estimated IMU accelerometer bias. The default value of 0.005 allows for up to 0.5 m/s/s of accelerometer bias.

COM_ARM_EKF_GB

The COM_ARM_EKF_GB parameter controls the maximum allowed EKF estimated IMU gyro bias. The default value of 0.00087 allows for up to 5 deg/sec of switch on gyro bias.

COM_ARM_IMU_ACC

The COM_ARM_IMU_ACC parameter controls the maximum allowed inconsistency in acceleration measurements between the default IMU used for flight control and other IMU units if fitted.

COM_ARM_IMU_GYR

The COM_ARM_IMU_GYR parameter controls the maximum allowed inconsistency in angular rate measurements between the default IMU used for flight control and other IMU units if fitted.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

仿真

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

软件在环仿真 (SITL)

官网英文原文地址：<http://dev.px4.io/simulation-sitl.html>

软件在环仿真是在主机上运行一个完整的系统并模拟自驾仪。它通过本地网络连接到仿真器。设置成如下的形式：



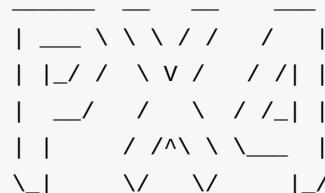
运行SITL

在确保**仿真必备条件** 已经安装在系统上之后，就可以直接启动：使用便捷的 `make target` 可以编译POSIX的主构建，并运行仿真。

```
make posix_sitl_default jmavsim
```

这将启动PX4 shell:

```
[init] shell id: 140735313310464
[init] task name: mainapp
```



```
Ready to fly.
```

```
pxh>
```

重要的文件

- 启动脚本文件在 `posix-configs/SITL/init` 文件夹中并被命名为 `rcs_SIM_AIRFRAME`，默认是 `rcs_jmavsim_iris`。
- 系统启动文件(相当于 `/` 被视为)位于构建文件夹内部：

```
build_posix_sitl_default/src/firmware posix/rootfs/
```

起飞

添加一个带[jMAVSim](#)仿真器的3D视觉窗口：



一旦完成初始化，该系统将打印home的位置 (`telem> home: 55.7533950, 37.6254270, -0.00`).
你能够通过输入以下指令让其起飞：

```
pxh> commander takeoff
```

提示 地面站(QGC)支持虚拟操纵杆或拇指操纵杆。要使用手动输入，把系统打在手动飞行模式(e.g. POSCTL, 位置控制)。从地面站QGC参考菜单启动拇指操纵杆。

Wifi无人机的仿真

现有一个特殊的任务：对通过局域网WiFi连接的无人机进行仿真

```
make broadcast jmavsim
```

如同一个真正的无人机会做的一样，仿真器也会广播无人机在局域网中的地址

扩展和自定义

为了扩展或自定义仿真界面，可以编辑 `Tools/jMAVSIM` 文件夹下的文件。能够通过 Github 上的 [jMAVSIM repository](#) 取得原码。

提示 构建系统强制检查所有依赖的子模块，包括仿真软件。虽然这些文件夹中文件的改变不会被覆盖，但当这些改变被提交的时候子模块需要在固件库中以新的hash注册。为此，输入 `git add Tools/jMAVSIM` 进行提交。这样仿真软件的GIT hash就会被更新。

连接ROS

这个仿真能够[连接到ROS上](#)，其方法与将一个搭载真实的飞行器的板子连接到ROS上相同。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

Gazebo 仿真

官网英文原文地址：<http://dev.px4.io/simulation-gazebo.html>

Gazebo 仿真

Gazebo是一个自主机器人3D仿真环境。它可以与ROS配套用于完整的机器人仿真，也可以单独使用。本文简要介绍单独的使用方法。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video



安装

需要安装Gazebo和我们的仿真插件。

提示 推荐使用Gazebo 7（最低使用Gazebo 6）。如果你的Linux操作系统安装的ROS版本早于Jade，请先卸载其绑定的旧版本Gazebo (`sudo apt-get remove ros-indigo-gazebo`)，因为该版本太老了。

Mac OS

Mac OS需要安装Gazebo 7，相应的需要安装xquartz，并且在没有OpenCV时无法运行。

```

brew cask install xquartz
brew install homebrew/science/opencv
brew install gazebo7
  
```

Linux

PX4 SITL使用Gazebo仿真软件，但不依赖ROS。但是也可以像普通飞行代码一样与ROS连接进行仿真。

ROS 用户

如果你计划与ROS一起用PX4，确保按照[Gazebo 7版本指南](#)进行配置。

正常安装

按照[Linux安装指导](#) 安装Gazebo 7。

确保gazebo7和libgazebo7-dev都装上了。

进行仿真

在PX4固件源文件的目录下运行一种机架类型（支持四旋翼、固定翼和垂直起降，含光流）的PX4 SITL。

注意：您可以使用下面的说明来保持Gazebo运行，并且只用重新启动PX4。

四旋翼

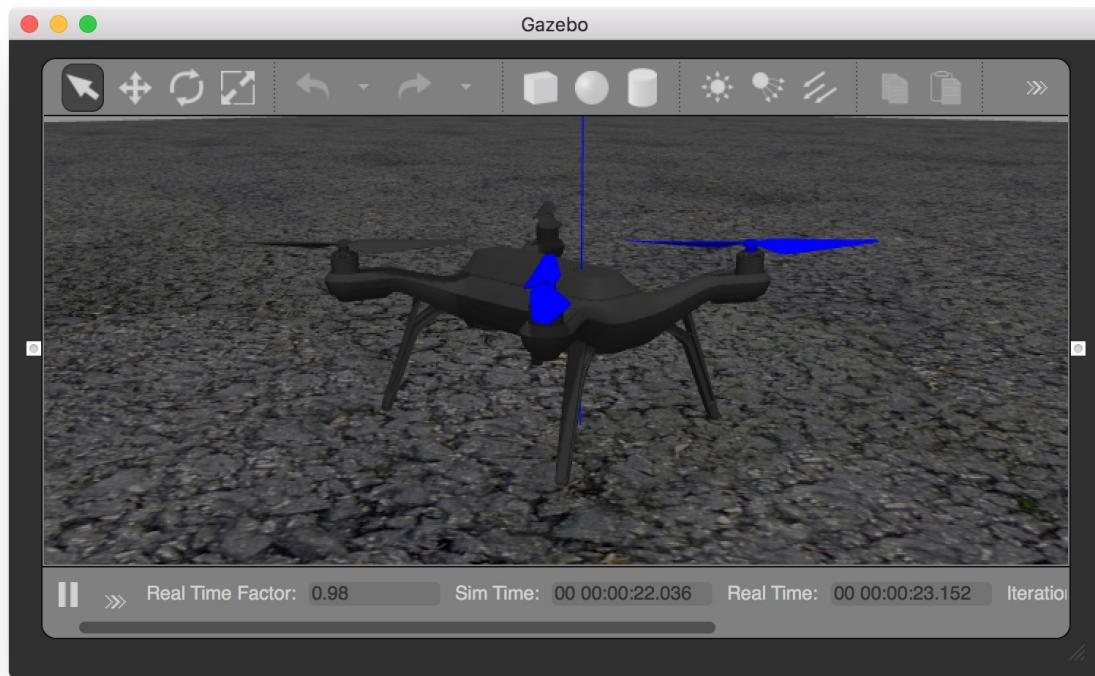
```
cd ~/src/Firmware  
make posix_sitl_default gazebo
```

四旋翼带光流模块

```
cd ~/src/Firmware  
make posix_gazebo_iris_opt_flow
```

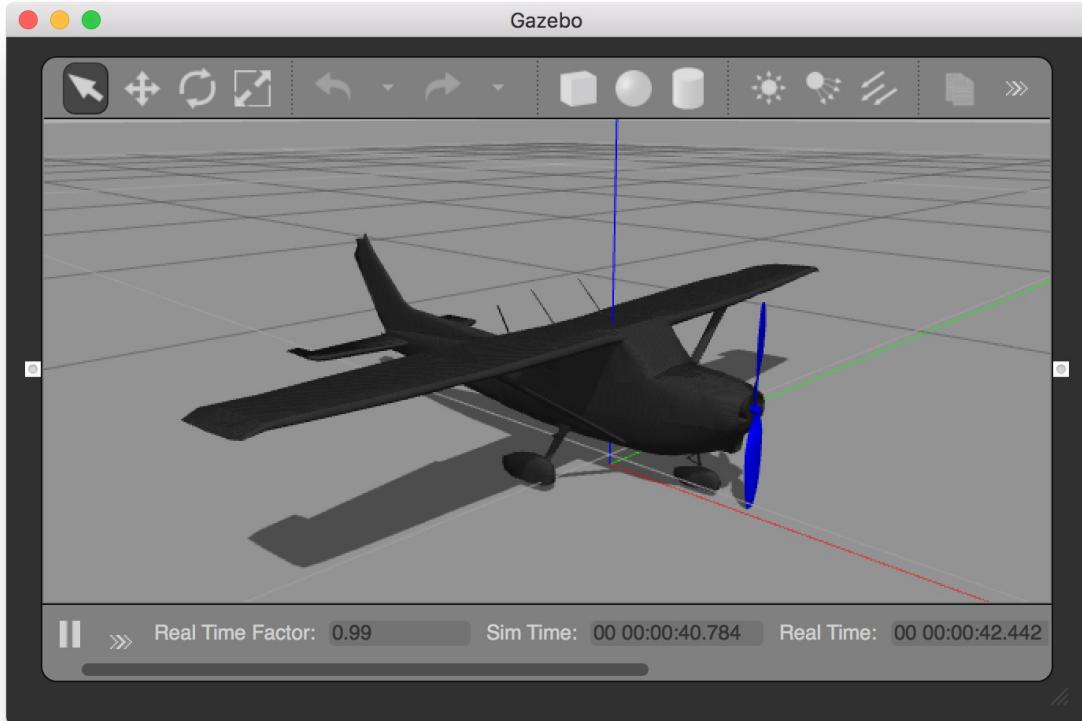
3DR Solo

```
make posix_gazebo_solo
```



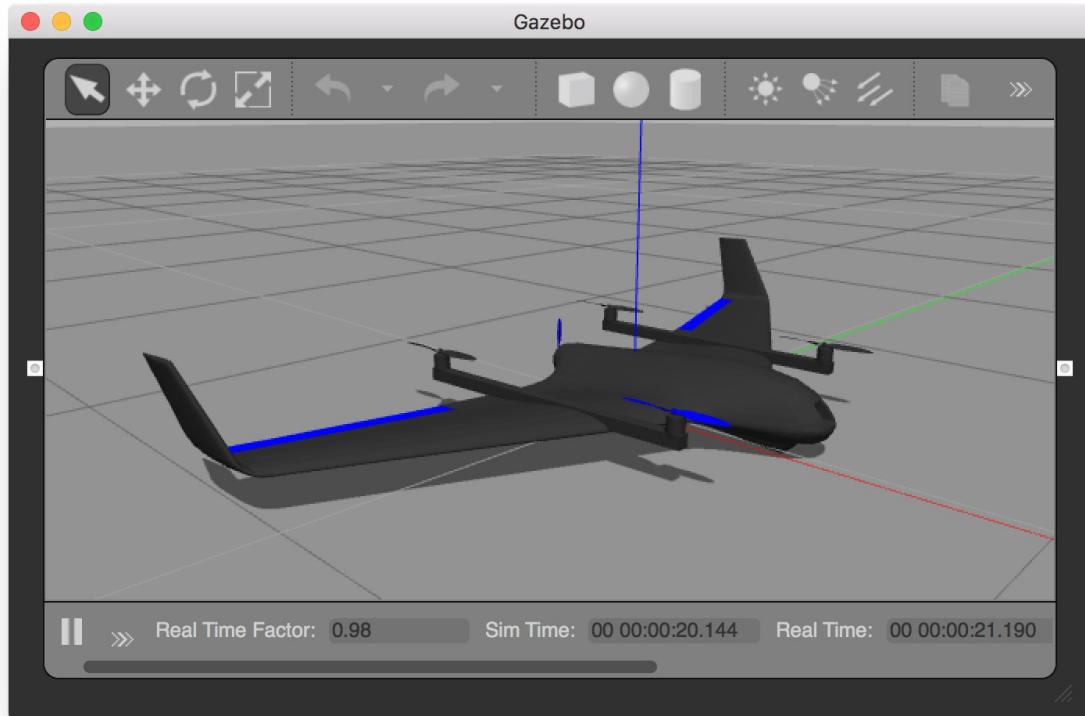
标准飞机

```
make posix gazebo_plane
```



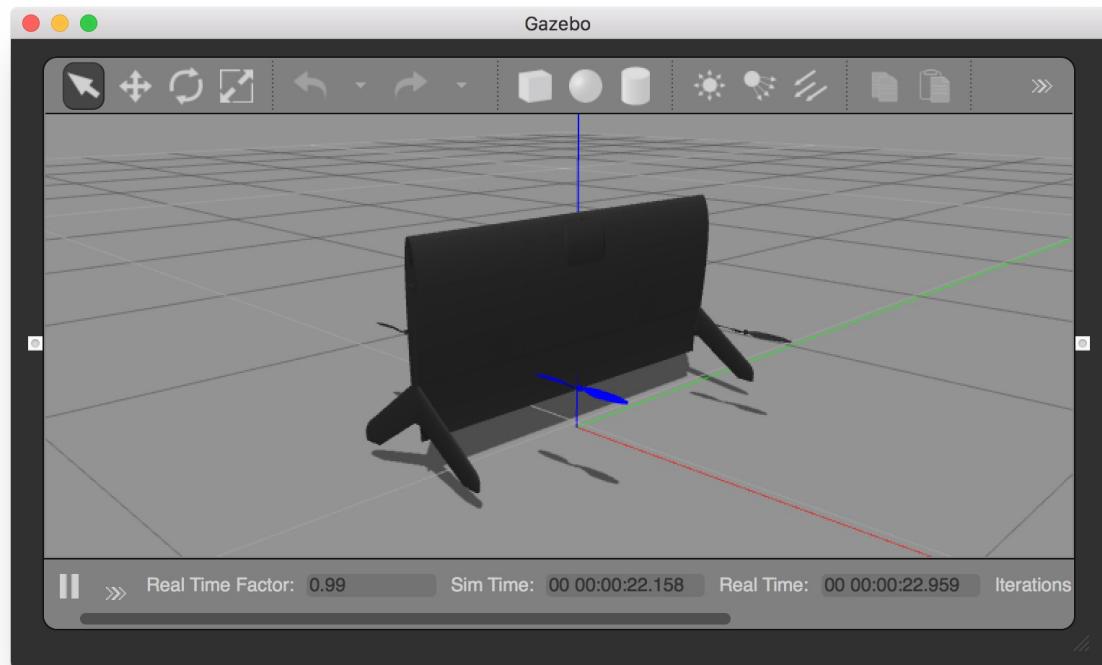
标准垂直起降飞机

```
cd ~/src/Firmware  
make posix_sitl_default  
gazebo_standard_vtol
```



立式垂直起降

```
cd ~/src/Firmware  
make posix_sitl_default gazebo_tailsitter
```



起飞

提示 如果你在运行的时候遇到错误或缺少依赖，确保你是按照[安装文件和代码](#)安装的。

接着会启动PX4 shell:

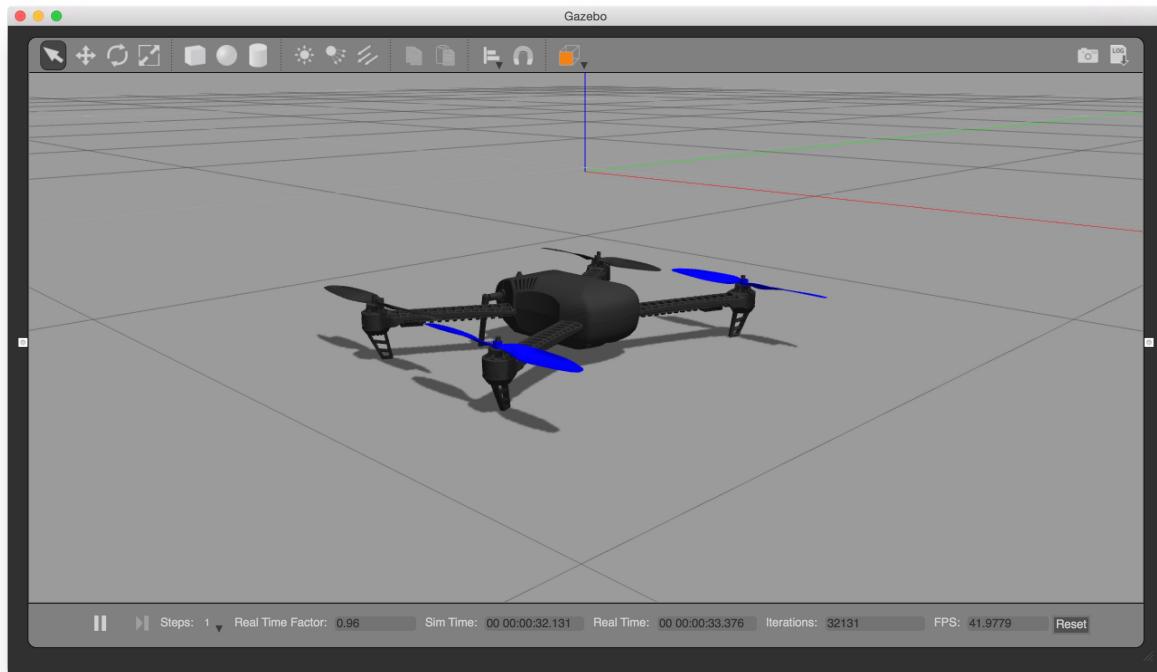
```
[init] shell id: 140735313310464
[init] task name: mainapp
```

```
____ \ \ \ / / / \ |
| _/ / \ v / / / | | |
| _/ / \ \ / / _ | |
| | / / ^ \ \ \ \_ | |
\_| \ \ \ \ \ \ / | / |
```

Ready to fly.

pxh>

右击四旋翼模型可以从弹出的菜单中启用跟随模式，这将会始终保持飞行器在视野中。



一旦完成初始化，系统将会打印出起始位置(`telem> home: 55.7533950, 37.6254270, -0.00`)。你可以通过输入下面的命令让飞行器起飞：

```
pxh> commander takeoff
```

提示 QGroundControl(QGC)支持手柄或拇指手柄。为了使用手柄控制飞行器，要将系统设为手动飞行模式（如 POSCTL，位置控制），并从QGC的选项菜单中启用拇指手柄。

单独启动Gazebo和PX4

对于扩展开发会话(development sessions)，单独启动Gazebo和PX4可能会更为方便，甚至还可以从IDE中启动。

除了现有的使用px4参数运行`sitl_run.sh`来加载正确的模型的cmake目标(target)之外，它还创建了一个名为px4_的启动器目标(launcher targets)(这是原始`sitl px4`应用程序的thin wrapper)。这个thin wrapper只是简单地嵌入应用程序参数，如当前工作目录和模型文件的路径。

如何使用

- 通过终端运行gazebo (或任何其他模拟器) 服务器(server)和客户端(client)查看器：

```
make posix_sitl_default gazebo_none_ide
```

- 在您的IDE中选择您要调试的px4_ 目标（例如 px4_iris ）
- 直接从IDE启动调试会话(session)

这种方法显着减少了调试周期时间，因为模拟器（例如Gazebo）总是在后台运行，并且您只用重新运行非常light的px4进程。

扩展和自定义

为了扩展和定制仿真接口，编辑 Tools/sitl_gazebo 文件夹中的文件。这些代码可以从Github上的[sitl_gazebo repository](#)访问。

提示 构建系统强制检查所有依赖的子模块，包括仿真软件。虽然这些文件夹中文件的改变不会被覆盖，但当这些改变被提交的时候子模块需要在固件库中以新的hash注册。为此，输入 `git add Tools/sitl_gazebo` 进行提交。这样仿真软件的GIT hash就会被更新。

与ROS连接

仿真可以像真实的飞控一样[与ROS连接](#)

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22
12:56:03

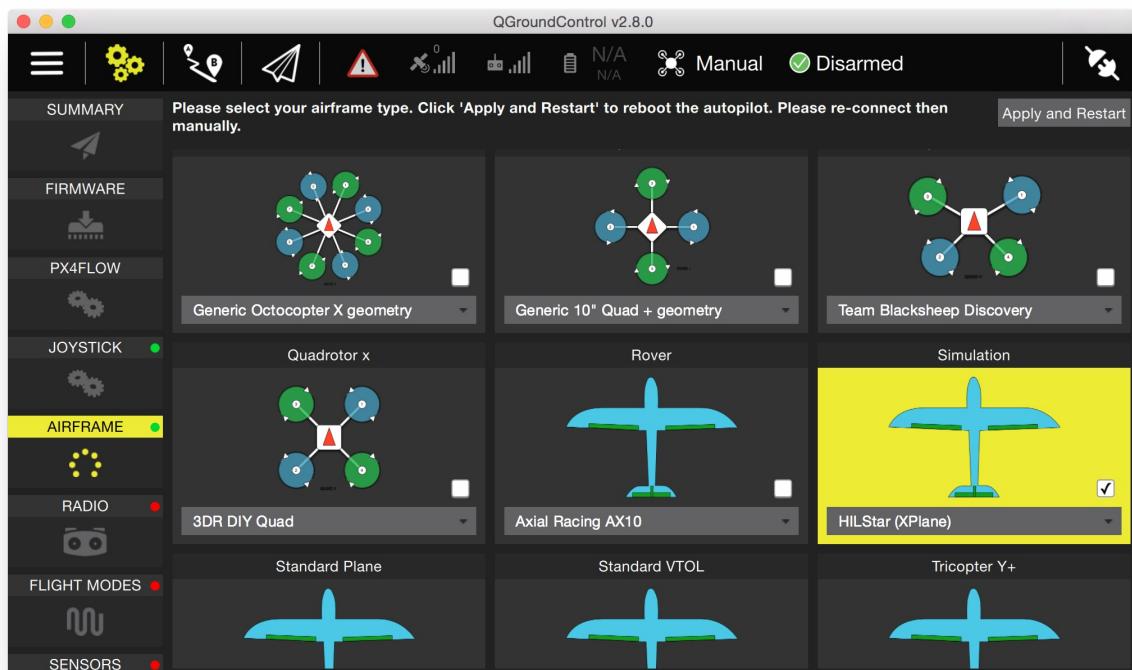
硬件在环仿真

官网英文原文地址：<http://dev.px4.io/simulation-hil.html>

硬件在环仿真指的自驾仪与仿真器相连并且所有的代码运行在自驾仪上的仿真。这种方法的优点是可以测试代码在实际处理器中的运行情况。

配置硬件在环仿真系统

PX4支持多旋翼（使用jMAVSIM）和固定翼（使用X-Plane demo或者full）的硬件在环仿真。虽然支持Flightgear，但是推荐使用X-Plane。通过机架菜单配置来使用硬件在环仿真。

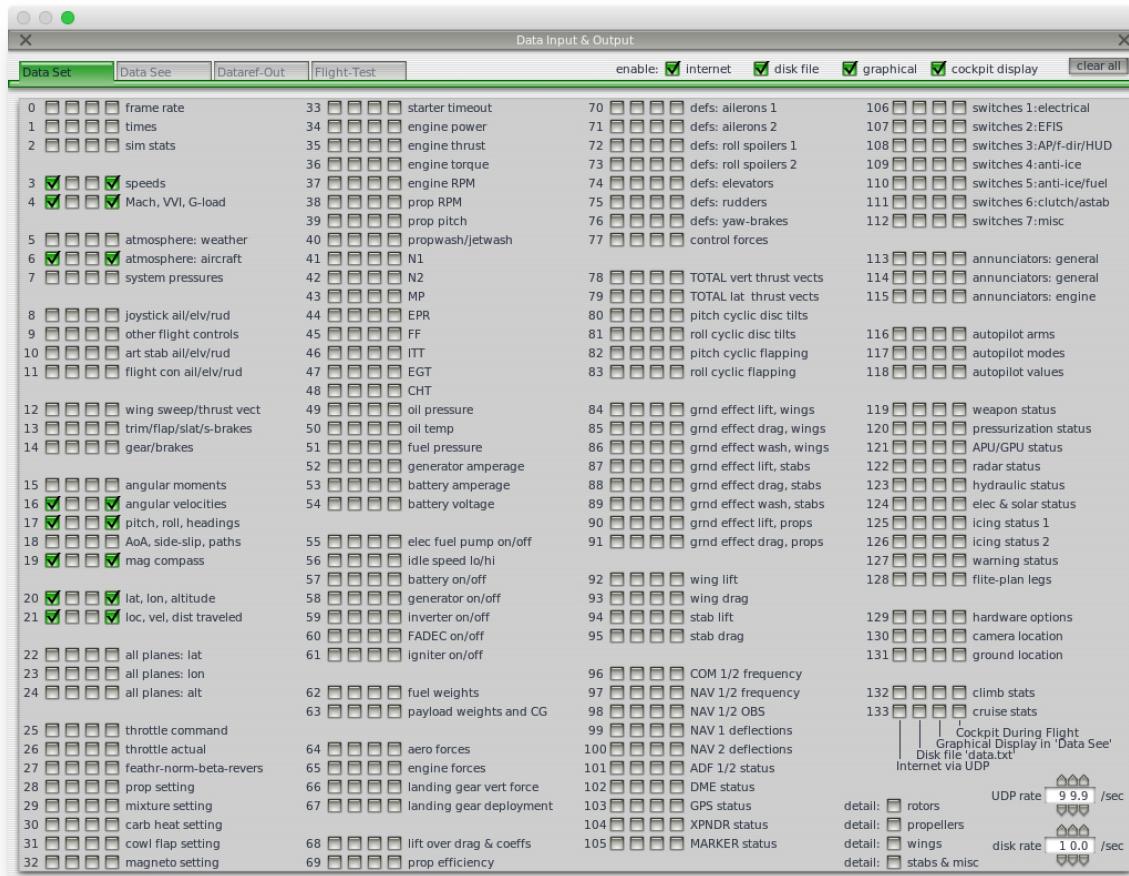


切换到Joystick输入

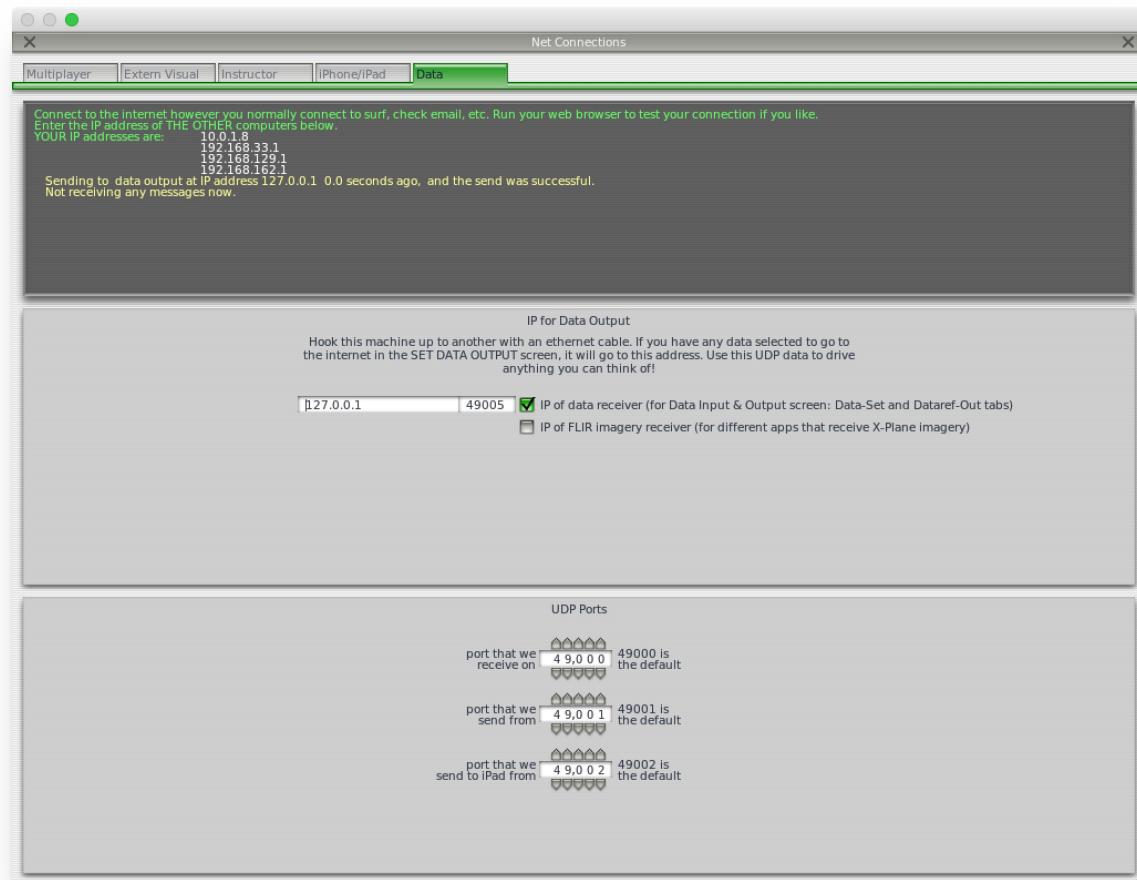
如果相比遥控器更喜欢使用joystick，那么可以设置参数 `COM_RC_IN_MODE` 为1来启用joystick。可以在Command参数组中找到这个参数。

允许远程访问X-Plane

在X-Plane中必须进行两项关键设置：在Settings -> Data Input and Output中，参照图中复选框设置：

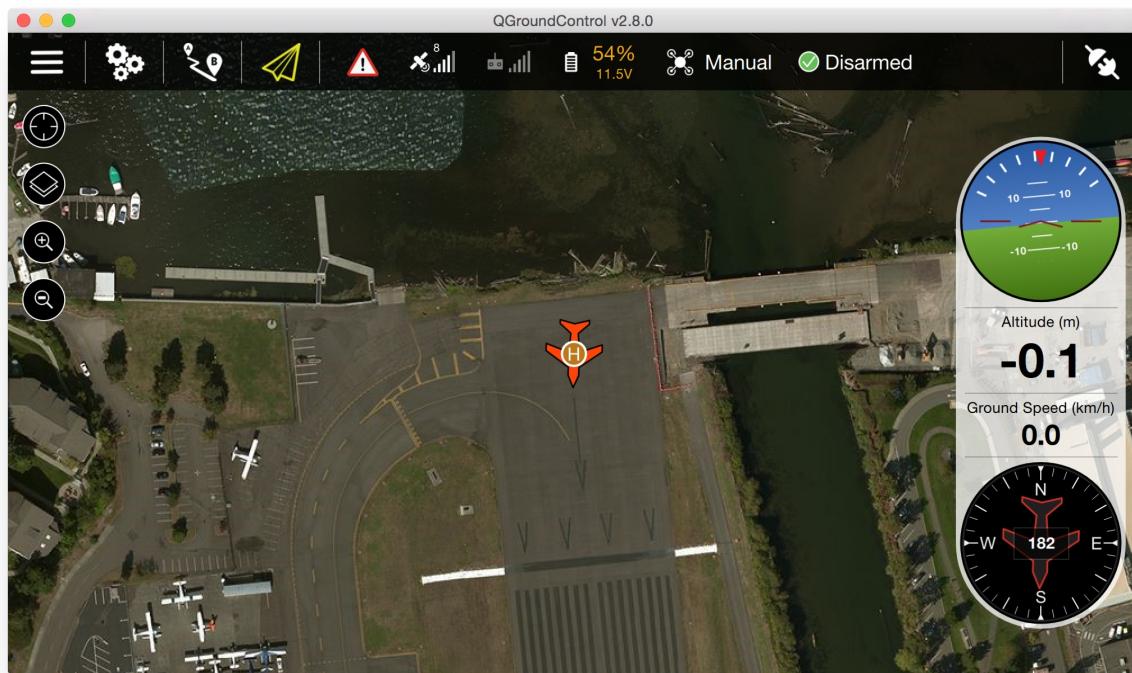


在Settings -> Net Connections的Data选项卡中，设置localhost以及端口49005作为IP地址，如下图所示：



在QGroundControl中启用硬件在环仿真

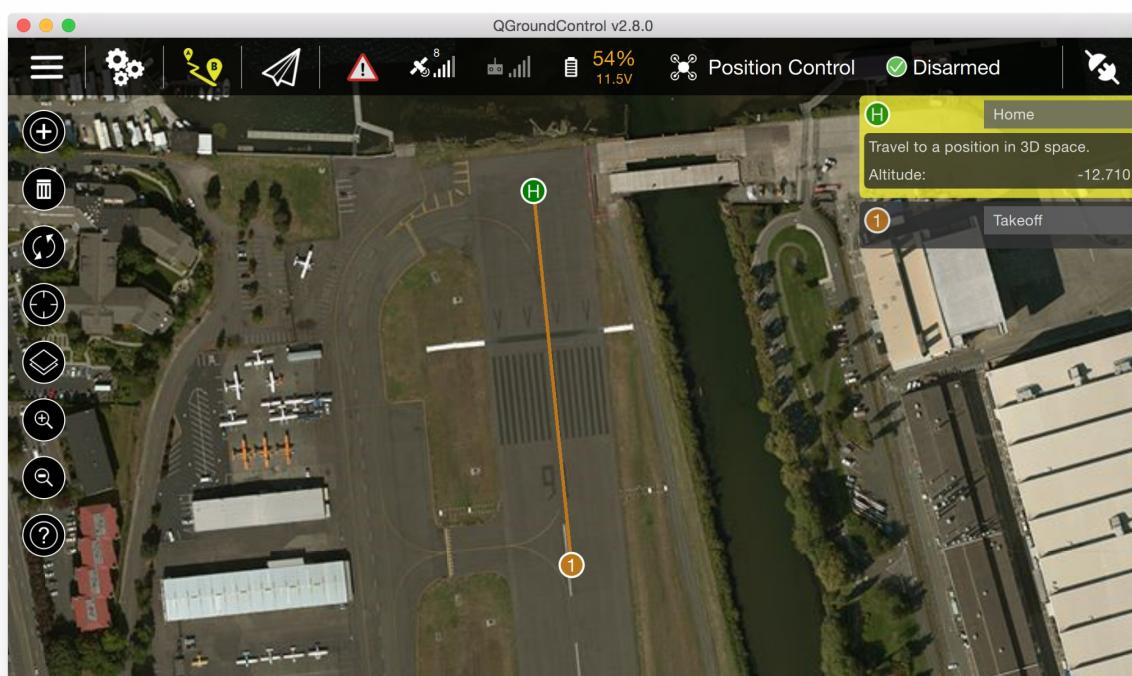
在Widgets -> HIL Config中，选中下拉菜单中的X-Plane 10，点击'connect'。一旦系统成功连接，电池状态，GPS状态和飞行器位置应该变为有效：



在硬件在环仿真中执行自动飞行任务

切换到flight planning页面，在飞机前面放置一个路径点。点击同步图标向自驾仪发送路径点。

接下来在工具栏的飞行模式菜单中选择MISSION模式，点击'DISARMED'解锁飞机。飞机将起飞并围绕起飞点盘旋。



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

ROS仿真接口

官网英文原文地址：<http://dev.px4.io/simulation-ros-interface.html>

模拟自驾仪会在端口14557开放第二个MAVLink接口。将MAVROS连接到这个端口将会接收到实际飞行中发出的所有数据。

启动MAVROS

如果需要ROS接口，那么已经在运行的次级(secondary)MAVLink实例可以通过mavros连接到ROS。若要连接到一个特定的IP(`fcu_url`是SITL的IP/端口)，请使用一个如下形式的URL：

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@192.168.1.36:14557"
```

若要链接到本地，请使用这种形式的URL：

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

为ROS安装Gazebo

Gazebo ROS SITL仿真可以在Gazebo 6和Gazebo 7上正常运行，可以通过如下方式安装：

```
sudo apt-get install ros-$(ROS_DISTRO)-gazebo7-ros-pkgs //Recommended
```

或者

```
sudo apt-get install ros-$(ROS_DISTRO)-gazebo6-ros-pkgs
```

使用ROS装饰器启动Gazebo

如果想要修改Gazebo仿真，使其能够将额外的传感器信息直接发布到ROS主题，例如Gazebo ROS激光传感器信息，那么必须通过适当的ROS包装器来启动Gazebo。

下面是一些可用的ROS启动脚本，这些脚本可以在ROS中运行包装过的仿真。

- `posix_sitl.launch`: 简单SITL

- `mavros posix sitl.launch`: SITL和MAVROS

为了在ROS中运行包装过的SITL，需要升级ROS环境，然后正常启动：

```
cd <Firmware_clone>
make posix_sitl_default
source ~/catkin_ws/devel/setup.bash
source Tools/setup_gazebo.bash $(pwd) build_posix_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
roslaunch px4 posix_sitl.launch
```

在自己的启动文件中包含上面提到的启动文件中的任意一个就可以在仿真中运行自己的ROS应用。

背后的细节

(或者如何手动运行)

```
no_sim=1 make posix_sitl_default gazebo
```

这将启动仿真器，控制台看上去是这样的：

现在，在一个新的终端中确保你可以通过Gazebo菜单插入Iris模型，这需要设置环境变量，在其中包含适当的 `sitl_gazebo` 文件夹。

```
cd <Firmware_clone>
source Tools/setup_gazebo.bash $(pwd) <Firmware_clone/build_posix_sitl_default>
```

这里 `Firmware_clone` 指的就是你下载Firmware时的文件夹.可以具体查看Tools下的 `README.md` 以及 `setup_gazebo.bash` 文件.

现在，就像在ROS中做过的那样启动Gazebo，并在其中插入Iris四旋翼模型。一旦Iris模型载入，它将会自动连接到px4应用。

```
roslaunch gazebo_ros empty_world.launch world_name:=$(pwd)/Tools/sitl_gazebo/worlds/iris.
```

如果遇到模型无法载入的情况,尝试将 `sitl_gazebo` 文件夹下的 `models` 中的模型复制到 `home` 文件夹下的 `.gazebo/models` 中.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

自驾仪的硬件

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Crazyflie 2.0

官网英文原文地址：<http://dev.px4.io/hardware-crazyflie2.html>

Crazyfile系列的微型四轴飞行器是由Bitcraze AB创建的。关于Crazyfile 2(CF2)的概况在[这里](#)。



简要概括

主要的硬件文档在[这里](#):

- 系统主芯片：STM32F405RG
 - CPU: 带单精度FPU的168 MHz ARM Cortex M4
 - RAM: 192KB SRAM
- nRF51822电台和电源管理控制器
- MPU9250加速度计/陀螺仪/磁力计
- LPS25H气压计

刷固件

设置好PX4开发环境之后，按照以下步骤可将PX4软件安装到CF2上：

1. 从GitHub上获取PX4 [Bootloader](#)的源码
2. 用 `make crazyflie_bl` 指令进行编译
3. 让CF2进入DFU模式
 - 保证其开始时未上电
 - 按住按键
 - 连接到电脑的USB口
 - 一秒后，蓝色的LED灯应该开始闪烁，五秒后应该闪的更快
 - 松开按键
4. 使用dfu-util刷Bootloader，输入指令

```
sudo dfu-util -d 0483:df11 -a 0 -s 0x08000000 -D crazyflie_bl.bin
```

完成后断开CF2。

- 如果成功的话，CF2再次连接时黄色的LED灯应该闪烁

5. 从GitHub上获取PX4 [Firmware](#)的源码
6. 用 `make crazyflie_default upload` 指令上传固件
7. 提示接入设备时，连接CF2:黄色的LED灯开始闪烁表明当前处于bootloader模式。然后红色的LED灯被点亮表明烧录过程已经开始了。
8. 等待完成
9. 完成！通过QGC地面站进行校准

无线

板载的nRF模块支持通过蓝牙或专有2.4ghz北欧ESB协议连接。

- 推荐使用[Crazyradio PA](#)
- 要立即试飞CF2，Crazyfile手机app支持通过蓝牙连接

使用官方的Bitcraze **Crazyflie**手机app

- 通过蓝牙连接
- 在设置中更改模式为1或者2
- 用QGC校准

通过**MAVLink**连接

- 使用Crazyfile PA，外加一个兼容的地面站
- 从[cfbridge](#)中查看如何将支持UDP的地面站连接到接收机上

飞行视频

To view this video please enable JavaScript, and consider upgrading to a web browser that supports [HTML5 video](#)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Intel Aero

官网英文原文地址：<http://dev.px4.io/hardware-intel-aero.html>

The Aero is a UAV development platform. Part of this is the Intel Aero Compute Board (see below), running Linux on a Quad-core CPU. This is connected to the FMU, which runs PX4 on NuttX.



Introduction

The main documentation is under <https://github.com/intel-aero/meta-intel-aero/wiki>. It includes instructions how to setup, update and connect to the board. And it also explains how to do development on the Linux side.

The following describes how to flash and connect to the FMU.

Flashing

After setting up the PX4 development environment, follow these steps to put the PX4 software on the FMU board:

1. Do a full update of all software on the Aero (<https://github.com/intel-aero/meta-intel-aero/wiki/Upgrade-To-Latest-Software-Release>)
2. Grab the [Firmware](#)
3. Compile with `make aerofc-v1_default`
4. Set the hostname (The following IP assumes you are connected via WiFi):

```
export AERO_HOSTNAME=192.168.1.1`
```

5. Upload with `make aerofc-v1_default upload`

Connecting QGroundControl via Network

1. Make sure you are connected to the board with WiFi or USB Network
2. ssh to the board and make sure mavlink forwarding runs. By default it automatically starts when booting. It can be started manually with:

```
/etc/init.d/mavlink_bridge.sh start
```

3. Start QGroundControl and it should automatically connect.
4. Instead of starting QGroundControl, you can open a [NuttX shell](#) with:

```
./Tools/mavlink_shell.py 0.0.0.0:14550
```

Pixfalcon硬件

官网英文原文地址：<http://dev.px4.io/hardware-pixfalcon.html>

Pixfalcon 是衍生于[Pixhawk](#)的设计，优化在空间受限的应用，如 FPV(First Person View, 第一人称主视角)，由[Holybro](#)设计。考虑到减小尺寸，**Pixfalcon**有更少的IO。对于性能要求较高或需要带有照相机的无人机，高通骁龙[Snapdragon Flight](#)可能会更加合适。



快速摘要

- 主片上系统: [STM32F437](#)
 - CPU: 180 MHz, ARM Cortex M4内核，单精度FPU
 - RAM: 256 KB SRAM
- 失效保护片上系统: STM32F100
 - CPU: 24 MHz ARM Cortex M3
 - RAM: 8 KB SRAM
- GPS: U-Blox M8 (捆绑)
- 光流: [PX4光流模块](#)，来自制造商[Holybro](#)或者分销商[Hobbyking](#)
- 配件: 来自制造商[Holybro](#)或者分销商[Hobbyking](#)
- 数字空速传感器：来自制造商[Holybro](#)或者分销商[Hobbyking](#)
- 集成测量显示在屏幕上:
 - [Hobbyking OSD + US 数传 \(915 MHz\)](#)
 - [Hobbyking OSD + EU 数传 \(433 MHz\)](#)
- 纯测量方案:
 - [Hobbyking Wifi 数传](#)
 - [Hobbyking EU 迷你数传\(433 MHz\)](#)

- Hobbyking US 迷你数传 (915 MHz)

接口

- 一个I2C
- 二个UART (一个给数传 / OSD, 没有光流控制)
 - 八通道 PWM 用来做手动操作
 - S.BUS / PPM 输入

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22

12:56:03

Pixhawk 硬件

官网英文原文地址：<http://dev.px4.io/hardware-pixhawk.html>

Pixhawk是PX4飞行堆栈的标准微控制器平台。它在**NuttX**操作系统上运行**PX4**中间件。依照CC-BY-SA 3.0 许可开放硬件设计，所有图纸和设计文件都是可获得的。**Pixfalcon**是**Pixhawk**的FPV 和类似平台较小版本。对于无人机性能要求较高或照相，高通骁龙**Snapdragon Flight**可能会更加合适。



快速摘要

- 主片上系统: [STM32F437](#)
 - CPU: 180 MHz, ARM Cortex M4内核，单精度FPU
 - RAM: 256 KB SRAM
- 失效保护片上系统: STM32F100
 - CPU: 24 MHz ARM Cortex M3
 - RAM: 8 KB SRAM
- Wifi: 外加ESP8266
- GPS: U-Blox 7/8 (Hobbyking) / U-Blox 6 (3D Robotics)
- 光流: [PX4 光流模块](#)
- 可购买:
 - [Hobbyking EU version \(433 MHz\)](#)
 - [Hobbyking US version \(915 MHz\)](#)
 - [3D Robotics Store](#)(GPS 和数传不捆绑)
- 配件:
 - [数字空速管](#)

- [Hobbyking Wifi 数传](#)
- [Hobbyking OSD + US 数传 \(915 MHz\)](#)
- [Hobbyking OSD + EU 数传 \(433 MHz\)](#)

接口

- 一个 I2C
- 一个 CAN (2x 可选)
 - 一个 ADC
 - 四个 UART (二个给光流控制)
 - 一个 Console 口
 - 八路 PWM 用来做手动控制
 - 六路 PWM / GPIO / PWM 输入
 - S.BUS / PPM / Spektrum 输入
 - S.BUS 输出

引出线和原理图

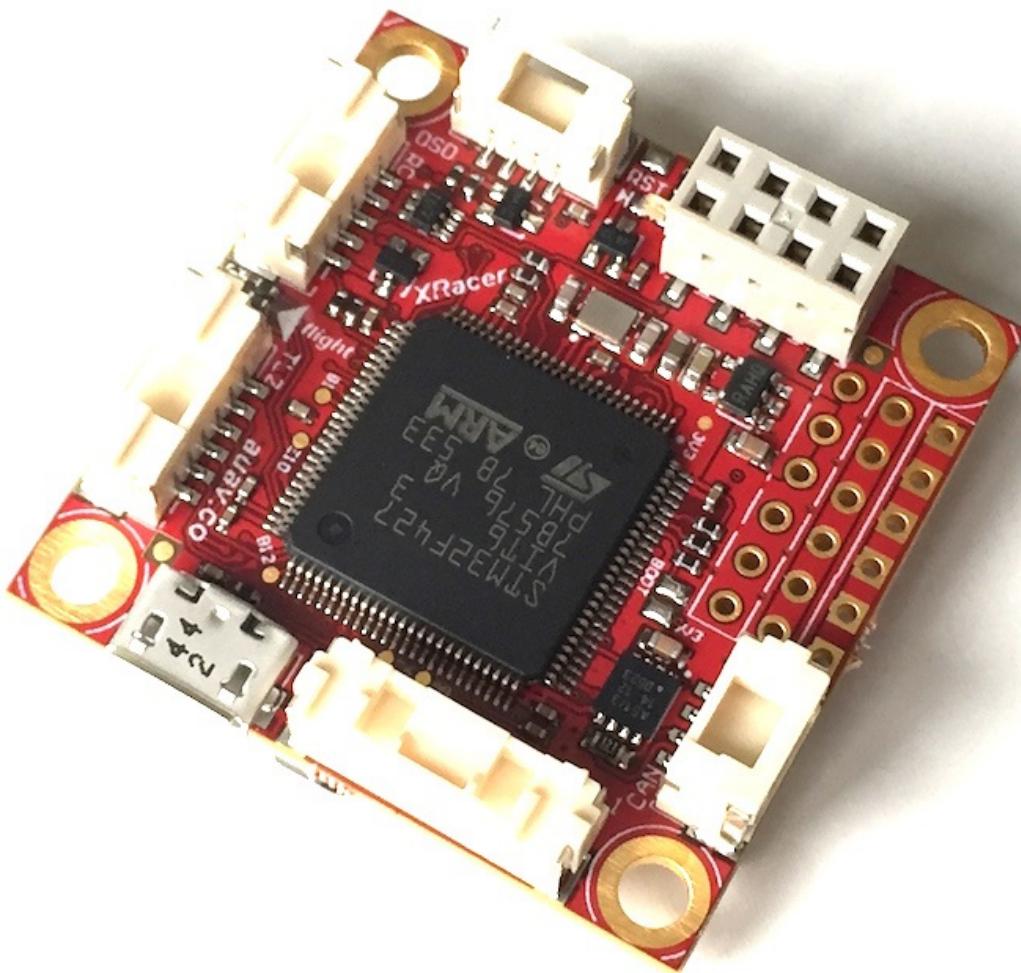
板子信息详细记录 [Pixhawk的项目网站](#)。

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Pixracer

官网英文原文地址：<http://dev.px4.io/hardware-pixracer.html>

Pixhawk XRacer板系列专门为小型赛车和飞机进行了优化。与 [Pixfalcon](#)和[Pixhawk](#)相反，
Pixracer具有内置 wifi、新传感器、方便的全速接口、CAN口、支持 2 M 闪存。



快速摘要

主要硬件文档在这里: <https://pixhawk.org/modules/pixracer>

- 主片上系统: [STM32F437](#)
 - CPU: 180 MHz, ARM Cortex M4内核，单精度FPU
 - RAM: 256 KB SRAM
- FPV标准参数: 36×36毫米，标准30.5毫米孔型

- Wifi 数传 and 软件升级
- 公司的 ICM-20608 加速度计 / 陀螺仪 (4 KHz) / MPU9250 加速度计 / 陀螺仪 / 磁力计 (4 KHz)
- HMC5983 磁力计的温度补偿
- Measurement Specialties公司的 MS5611 气压计
- JST GH 连接器
- microSD (记录日志)
- S.BUS / Spektrum / SUMD / PPM 输入
- FrSky 数传口
- OneShot PWM 输出 (可配置)
- 可选：安全开关和蜂鸣器
- 可购买：
 - [AUAV Pixracer](#)
- 配件：
 - [数字空速管](#)
 - [Hobbyking OSD + US 数传 \(915 MHz\)](#)
 - [Hobbyking OSD + EU 数传 \(433 MHz\)](#)

套件

Pixracer有专门设计的单独航空电子设备的电源。这是必要的，以避免浪涌电流从电机和电调回流到飞控，扰乱敏感的传感器。

- 电源模块（电压和电流监测）
- I2C扩展器（支持 AUAV，Hobbyking和3DR外设）
 - 常见的所有外围设备的电缆配套件

Wifi (无需USB)

板子的主要特征之一是它能够使用 Wifi 闪烁的新固件、系统设置和空中遥测。这将释放它的任何桌面系统的需要

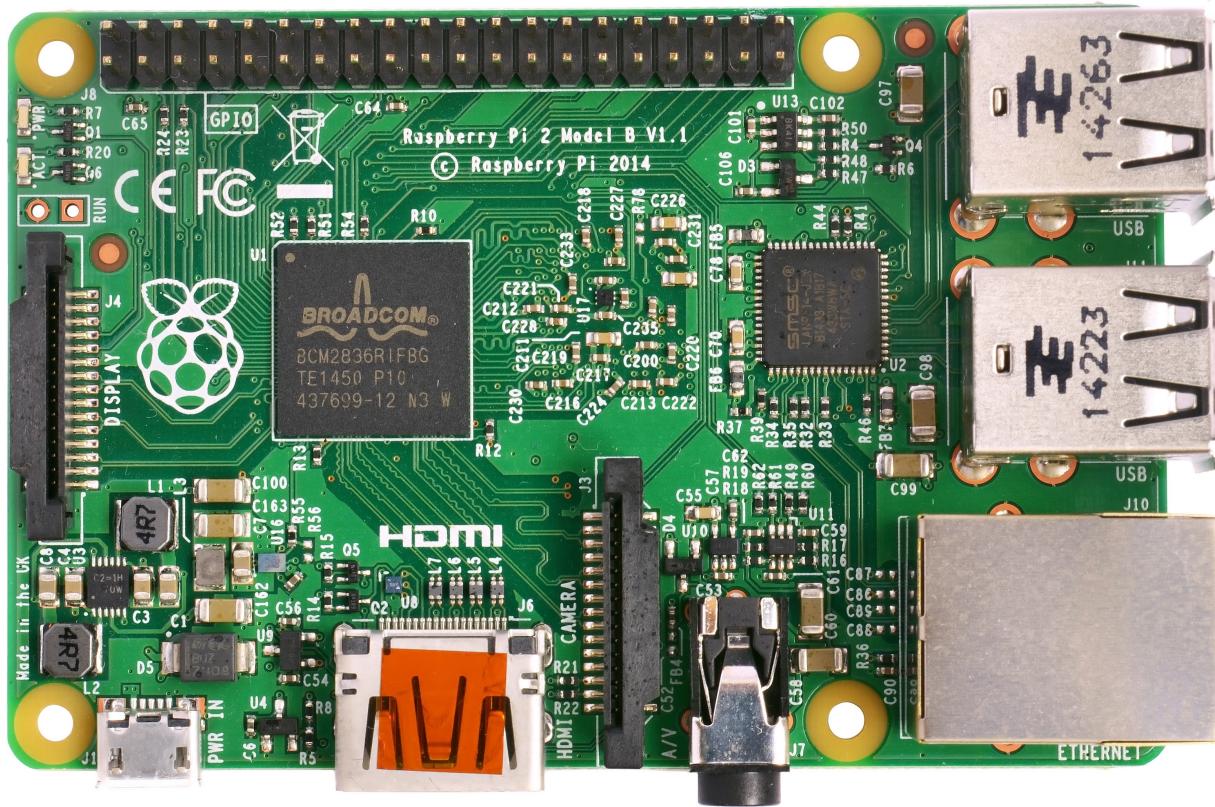
步骤和数传已经可用，固件升级已经由默认引导支持，但尚未启用。

- [ESP8266文档和Flash说明](#)
- [自定义 ESP8266 MAVLink firmware](#)[Pixracer](#)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

树莓派2/3自动驾驶仪

官网英文原文地址：<http://dev.px4.io/hardware-rpi.html>



开发者快速开始使用

OS Image 系统镜像

使用[Emlid RT Raspbian image](#)这个前期配置好的有效的PX4树莓派镜像。这个镜像默认最大化的事先配置了程序。

使用设置

此树莓派镜像已经事先设置了SSH。用户名：pi 和密码：raspberry。你可以直接通过网络去连接你的树莓派2（以太网已经启动并且默认自动分配IP）和可以配置使用wifi。在这篇文章档中，我们采取默认的用户名和密码登入树莓派系统。

设置你的树莓派加入你的本地wifi网络（使你的树莓派连接你的wifi），关于如何设置请参考这篇[指导文章](#)。找到树莓派在你网络中的IP地址，随后你可以开始通过SSH的方式连接Pi2。

```
ssh pi@<IP-ADDRESS>
```

改变树莓派主机名

为了避免与同一网络上的其他树莓派有冲突，我们建议你改变默认主机名为明显的名字。我们使用px4autopilot作为我们的主机名。通过ssh连接pi并执行指令。

编辑主机名文件：

```
sudo nano /etc/hostname
```

更改主机名 `raspberry` 为你想要的任何主机名 (一个word字的有限字符)

下一步更改主机host文件：

```
sudo nano /etc/hosts
```

更改入口 `127.0.1.1 raspberry` 为 `127.0.1.1 <YOURNEWHOSTNAME>` 做完这步重启电脑成功后允许他重新关联你的网络。

Setting up Avahi (Zeroconf)

为了使你更容易的连接上你的树莓派，我们推荐设置Avahi（Zeroconf：零配置网络服务规范）来更简单的使用连接网络。使连接到树莓派更容易，我们建议设立的avahi（zeroconf）允许方便地访问任何网络PI直接指定主机名。

```
sudo apt-get install avahi-daemon
sudo insserv avahi-daemon
```

接下来，设置Avahi配置文件

```
sudo nano /etc/avahi/services/multiple.service
```

把下面文字加入文件中：

```

<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
    <name replace-wildcards="yes">%h</name>
    <service>
        <type>_device-info._tcp</type>
        <port>0</port>
        <txt-record>model=RackMac</txt-record>
    </service>
    <service>
        <type>_ssh._tcp</type>
        <port>22</port>
    </service>
</service-group>

```

重新启动后台进程

```
sudo /etc/init.d/avahi-daemon restart
```

到此就结束了。你应该可以在同一网络上的任意一台电脑上通过树莓派的主机名访问你的树莓派。

You might have to add .local to the hostname to discover it.

Configuring a SSH Public-Key

为了允许PX4开发环境自动的更新可执行文件到你的开发板子上，你需要配置无密码访问RPi。我们使用公钥的验证方法。

输入下面的命令来创建新的SSH密钥 (使用合理的主机名如 <YOURNAME>@<YOURDEVICE> . 在这我们使用 pi@px4autopilot)

这些命令需要运行在主机开发电脑上！

```
ssh-keygen -t rsa -C pi@px4autopilot
```

执行上面这条命令，将会询问你这个密钥存在哪个位置。我们建议你直接回车让它存储在默认位置(\$HOME/.ssh/id_rsa)。

现在你可以看到这些文件 id_rsa 和 id_rsa.pub 在你的电脑主目录文件夹下的 .ssh 文件夹里：

```
ls ~/.ssh
authorized_keys  id_rsa  id_rsa.pub  known_hosts
```

这 `id_rsa` 文件是你的私人密钥. 让它保存在开发主机电脑上. 这 `id_rsa.pub` 文件是你的公共密钥. This is what you put on the targets you want to connect to.

把你的公共密钥复制到你的树莓派上，使用下面的命令来将你的公共密钥添加到树莓派上的密钥授权文件里。sending it over SSH:

```
cat ~/.ssh/id_rsa.pub | ssh pi@px4autopilot 'cat >> .ssh/authorized_keys'
```

注意：这一次你将通过密码的方式进行身份验证(默认是"raspberry").

现在尝试使用 `ssh pi@px4autopilot` 连接时没有密码提示.

如果你看到这条信息" Agent admitted failure to sign using the key. " 然后加入你的RSA或者DSA认证身份到你的 `ssh-agent` (密钥管理器) ，并执行下面的命令：

```
ssh-add
```

如果它不工作, 使用 `rm ~/.ssh/id*` 这条命令删除你的密钥，然后按照上面的操作重新再做一遍。

测试文件传输

我们使用SCP命令通过网络（wifi或者以太网）从开发主机传输文件到目标板。

为了测试你的设置, 现在尝试通过网络从开发主机上推送到树莓派上。确保你的树莓派能正确访问网络，并且你可以使用ssh访问它。

```
echo "Hello" > hello.txt
scp hello.txt pi@px4autopilot:/home/pi/
rm hello.txt
```

这是拷贝一个"hello.txt"文件到你树莓派的家目录下（/home/pi）。确认这个文件是真的拷贝进去了，你就可以执行下一步了。

本地构建 (可选)

如果你想你也可以直接在树莓派上进行PX4的构建。这就是本地构建目标。

其他方法就是通过交叉编译的方式在一台开发主机上为树莓派运行构建可执行目标，然后直接发送PX4可执行二进制文件到树莓派上。推荐使用交叉编译的方法，因为它可以加快开发和容易开发。对于使用交叉编译环境，你就可以跳过这一步。

这下面的安装脚本可以自动的在Pi上更新PX4的本地开发环境。在树莓派上运行这些命令。

```
git clone https://github.com/pixhawk/rpi_toolchain.git  
cd rpi_toolchain  
chmod +x install_native.sh  
../install_native.sh
```

Building the code

Continue with our [standard build system installation](#).

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

骁龙自动驾驶仪

官网英文原文地址：<http://dev.px4.io/hardware-snapdragon.html>

骁龙Snapdragon Flight平台是高端的自动驾驶仪 / 机载计算机，它的DSP上有QuRT实时操作系统来运行PX4，使用DSPAL API兼容与POSIX。与Pixhawk相比它添加了一个摄像头和WiFi及其高端的处理能力和不同的IO接口。

有关的骁龙平台飞行的更多信息在[Snapdragon-Flight-Details](#)



快速摘要

- 片上系统: [骁龙 801](#)
 - CPU: Quad-core 2.26 GHz Krait
- DSP: Hexagon DSP (QDSP6 V5A) – 801 MHz+256KL2 (运行飞控代码)
- GPU: Qualcomm® Adreno™ 330 GPU
- RAM: 2GB LPDDR3 PoP @931 MHz
- 内存: 32GB eMMC Flash
 - 录像: 索尼 IMX135 on Liteon Module 12P1BAD11
- 4k@30fps 3840×2160 video capture to SD card with H.264 @ 100Mbits (1080p/60 with parallel FPV), 720p FPV
- 光流: Omnidvision OV7251 on Sunny Module MD102A-200
 - 640x480 @ 30/60/90 fps
- Wifi: Qualcomm® VIVE™ 1-stream 802.11n/ac with MU-MIMO + Integrated digital core

- BT/WiFi: BT 4.0 and 2G/5G WiFi via QCA6234

- 802.11n, 2×2 MIMO with 2 uCOAX connectors on-board for connection to external antenna

- GPS: Telit Jupiter SE868 V2 module (使用外部UBLOX模块，建议由PX4代替)

- uCOAX 连接器用于连接到外部 GPS 车载贴片天线

- CSR SiRFstarV @ 5Hz via UART

- 加速度计 / 陀螺仪 / 磁力计: Invensense公司的MPU-9250 9-轴传感器, 3x3mm QFN, 接在 SPI1

- 气压计: Bosch公司的 BMP280 气压传感器, 接在 I2C3

- 电压: 5V 直流电用外部2S-6S电池通过APM适配器调节至5V

- 可购买: [Intrinsyc商店](#)

接口

- 一个USB 3.0 高速口 (micro-A/B)
- 微型 SD 卡插槽 - 万用接头 (PWB/GND/BLSP)
 - 电调接口 (2W UART)
 - I2C
 - 60针高速的Samtec QSH-030-01-L-D-A-K扩展连接器
 - 2x BLSP ([BAM 低速外设](#))
- USB

外置引线

尽管晓龙使用 DF13 连接器，外置引线还是有别于 Pixhawk。

WiFi

- WLAN0, WLAN1 (+BT 4.0): U.FL connector: [Taoglas 胶粘剂的天线 \(DigiKey公司\)](#)

连接器

串口的默认对应如下所示：

设备	描述
/dev/tty-1	J15 (next to USB)
/dev/tty-2	J13 (next to power module connector)
/dev/tty-3	J12 (next to J13)
/dev/tty-4	J9 (next to J15)

为自定义的 UART 到 BAM 映射，创建一个名为 "blsp.config" 文件和执行命令 adb push 把他下到 /usr/share/data/adsp。例如，保留默认的映射，你 "blsp.config" 应该看起来像: tty-1 bam-9

tty-2 bam-8

tty-3 bam-6

tty-4 bam-2

J9 / GPS

Pin	2-wire UART + I2C	SPI	Comment
1	3.3V	3.3V	3.3V
2	UART2_TX	SPI2_MOSI	Output (3.3V)
3	UART2_RX	SPI2_MISO	Input (3.3V)
4	I2C2_SDA	SPI2_CS	(3.3V)
5	GND	GND	
6	I2C2_SCL	SPI2_CLK	(3.3V)

J12 / 云台总线

Pin	2-wire UART + GPIO	SPI	Comment
1	3.3V	3.3V	
2	UART8_TX	SPI8_MOSI	Output (3.3V)
3	UART8_RX	SPI8_MISO	Input (3.3V)
4	APQ_GPIO_47	SPI8_CS	(3.3V)
5	GND	GND	
6	APQ_GPIO_48	SPI8_CLK	(3.3V)

J13 / 电调总线

Pin	2-wire UART + GPIO	SPI	Comment
1	5V	5V	
2	UART6_TX	SPI6_MOSI	Output (5V)
3	UART6_RX	SPI6_MISO	Input (5V)
4	APQ_GPIO_29	SPI6_CS	(5V)
5	GND	GND	
6	APQ_GPIO_30	SPI6_CLK	(5V)

J14 / 电源

Pin	Signal	Comment
1	5V DC	Power input
2	GND	
3	I2C3_SCL	(5V)
4	I2C3_SDA	(5V)

J15 / 无线接收器/传感器

Pin	2-wire UART + I2C	SPI	Comment
1	3.3V	3.3V	
2	UART9_TX	SPI9_MOSI	Output
3	UART9_RX	SPI9_MISO	Input
4	I2C9_SDA	SPI9_CS	
5	GND	GND	
6	I2C9_SCL	SPI9_CLK	

外设

UART to Pixracer / Pixfalcon 接线

这个接口是用来利用Pixracer/ Pixfalcon作为I/O接口板。连接到 `TELEM1` 在 Pixfalcon和在 `TELEM2` 在 Pixracer。

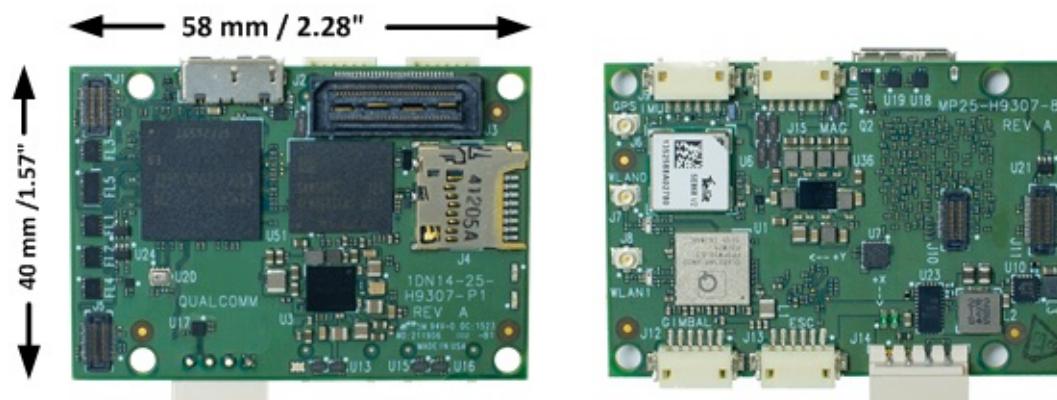
Snapdragon J13 Pin	Signal	Comment	Pixfalcon / Pixracer Pin
1	5V	Power for autopilot	5V
2	UART6_TX	Output (5V) TX -> RX	3
3	UART6_RX	Input (5V) RX -> TX	2
4	APQ_GPIO_29	(5V)	Not connected
5	GND		6
6	APQ_GPIO_30	(5V)	Not connected

GPS 接线

尽管3DR GPS要求5V输入，采用3.3V输入似乎也能很好地工作。（内置的调节器MIC5205具有2.5V的最小工作电压）。

Snapdragon J9 Pin	Signal	Comment	3DR GPS 6pin/4pin	Pixfalcon GPS pin
1	3.3V	(3.3V)	1	4
2	UART2_TX	Output (3.3V)	2/-	3
3	UART2_RX	Input (3.3V)	3/-	2
4	I2C2_SDA	(3.3V)	-/3	5
5	GND		6/-	1
6	I2C2_SCL	(3.3V)	-/2	6

尺寸



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

骁龙Flight的光流

官网英文原文地址：http://dev.px4.io/optical_flow.html

骁龙Flight有一个向下的灰度摄像头可用于实现基于光流的定位。

除了一个摄像头，光流还需要一个向下的距离传感器。这里讨论关于TeraRanger One的使用。

安装TeraRanger One

为了将TeraRanger One(TROne)连接到骁龙Flight上，必须使用TROne I2C适配器。TROne必须先由供应商刷好I2C固件。

TROne通过一根定制的DF13 4转6接口线与骁龙Flight连接。线序如下：

4 pin	<->	6 pin
1		1
2		6
3		4
4		5

TROne必须用10-20V的电压供电。

光流

光流获取的数据在应用处理器上进行计算并通过Mavlink传到PX4中。从GitHub上克隆[snap_cam](#)仓库的固件并根据README文件中的教程完成编译。

运行光流的程序来root：

```
optical_flow -n 50 -f 30
```

光流的程序需要来自PX4的IMU MavLink消息。你可能需要通过添加下列代码到你的 `mainapp.config` 中来添加一个MavLink例程到PX4中。

```
mavlink start -u 14557 -r 1000000 -t 127.0.0.1 -o 14558
mavlink stream -u 14557 -s HIGHRES_IMU -r 250
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Snapdragon Advanced

官网英文原文地址：<http://dev.px4.io/advanced-snapdragon.html>

Connect to Snapdragon

Over FTDI

Connect the small debug header shipped with the Snapdragon and the FTDI cable.

On Linux, open a console using:

```
screen /dev/ttyUSB0 115200
```

Change USB0 to whatever it happens to be. Check `/dev/` or `/dev/serial/by-id`.

Over ADB (Android Debug Bridge)

Connect the Snapdragon over USB2.0 and power it up using the power module. When the Snapdragon is running the, the LED will be slowly blinking (breathing) in blue.

Make sure the board can be found using adb:

```
adb devices
```

If you cannot see the device, it is most likely a USB device permission issue. Follow the instructions

To get a shell, do:

```
adb shell
```

Upgrade Snapdragon

For this step the Flight_BSP zip file from Intrynsic is required. It can be obtained after registering using the board serial.

Upgrading/replacing the Linux image

Flashing the Linux image will erase everything on the Snapdragon. Back up your work before you perform this step!

Make sure the board can be found using adb:

```
adb devices
```

Then, reboot it into the fastboot bootloader:

```
adb reboot bootloader
```

Make sure the board can be found using fastboot:

```
fastboot devices
```

Download the latest BSP from Intrinsyc:

```
unzip Flight_3.1.1_BSP_apq8074-00003.zip
cd BSP/binaries/Flight_BSP_4.0
./fastboot-all.sh
```

It is normal that the partitions `recovery`, `update`, and `factory` will fail.

Updating the ADSP firmware

Part of the PX4 stack is running on the ADSP (the DSP side of the Snapdragon 8074). The underlying operating system QURT needs to be updated separately.

If anything goes wrong during the ADSP firmware update, your Snapdragon can get bricked! Follow the steps below carefully which should prevent bricking in most cases.
First of all, if you're not already on BSP 3.1.1, [upgrade the Linux image](#)!

Prevent bricking

To prevent the system from hanging on boot because of anything wrong with the ADSP firmware, do the following changes before updating:

Edit the file directly on the Snapdragon over `screen` or `adb shell`:

```
vim /usr/local/qr-linux/q6-admin.sh
```

Or load the file locally and edit it there with the editor of your choice:

To do this, load the file locally:

```
adb pull /usr/local/qr-linux/q6-admin.sh
```

Edit it:

```
gedit q6-admin.sh
```

And push it back:

```
adb push q6-admin.sh /usr/local/qr-linux/q6-admin.sh
adb shell chmod +x /usr/local/qr-linux/q6-admin.sh
```

Comment out the while loops causing boot to hang:

```
# Wait for adsp.mdt to show up
#while [ ! -s /lib/firmware/adsp.mdt ]; do
#  sleep 0.1
#done
```

and:

```
# Don't leave until ADSP is up
#while [ "`cat /sys/kernel/debug/msm_subsys/adsp`" != "2" ]; do
#  sleep 0.1
#done
```

Push the latest ADSP firmware files

Download the file [Flight_3.1.1a_qcom_flight_controller_hexagon_sdk_add_on.zip](#) from Intrinsyc.

And copy them on to the Snapdragon:

```
unzip Flight_3.1.1a_qcom_flight_controller_hexagon_sdk_add_on.zip
cd images/8074-eagle/normal/adsp_proc/obj/qdsp6v5_ReleaseG/LA/system/etc/firmware
adb push . /lib/firmware
```

Then do a graceful reboot, so that the firmware gets applied:

```
adb reboot
```

Serial ports

Use serial ports

Not all POSIX calls are currently supported on QURT. Therefore, some custom ioctl are needed.

The APIs to set up and use the UART are described in [dspal](#).

Wifi-settings

These are notes for advanced developers.

Connect to the Linux shell (see [console instructions](#)).

Access point mode

If you want the Snapdragon to be a wifi access point (AP mode), edit the file:

`/etc/hostapd.conf` and set:

```
ssid=EnterYourSSID
wpa_passphrase=EnterYourPassphrase
```

Then configure AP mode:

```
/usr/local/qr-linux/wificonfig.sh -s softap
reboot
```

Station mode

If you want the Snapdragon to connect to your existing wifi, edit the file:

`/etc/wpa_supplicant/wpa_supplicant.conf` and add your network settings:

```
network={
    ssid="my existing network ssid"
    psk="my existing password"
}
```

Then configure station mode:

```
/usr/local/qr-linux/wificonfig.sh -s station  
reboot
```

Troubleshooting

adb does not work

- Check [permissions](#)
- Make sure you are using a working Micro-USB cable.
- Try a USB 2.0 port.
- Try front and back ports of your computer.

USB permissions

1) Create a new permissions file

```
sudo -i gedit /etc/udev/rules.d/51-android.rules
```

paste this content, which enables most known devices for ADB access:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bb4", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0e79", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0502", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0b05", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="413c", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0489", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="091e", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="18d1", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bb4", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="12d1", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="24e3", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2116", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0482", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="17ef", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1004", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="22b8", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0409", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2080", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0955", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="2257", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="10a9", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1d4d", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0471", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="04da", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="05c6", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1f53", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="04e8", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="04dd", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0fce", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0930", MODE="0666", GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="19d2", MODE="0666", GROUP="plugdev"
```

Set up the right permissions for the file:

```
sudo chmod a+r /etc/udev/rules.d/51-android.rules
```

Restart the deamon

```
sudo udevadm control --reload-rules
sudo service udev restart
sudo udevadm trigger
```

If it still doesn't work, check [this answer on StackOverflow](#).

Board doesn't start / is boot-looping / is bricked

If you can still connect to the board using the serial console and get to a prompt such as:

```
root@linaro-developer:~#
```

You can get into fastboot (bootloader) mode by entering:

```
reboot2fastboot
```

If the serial console is not possible, you can try to connect the Micro USB cable, and enter:

```
adb wait-for-device && adb reboot bootloader
```

Then power cycle the board. If you're lucky, adb manages to connect briefly and can send the board into fastboot.

To check if it's in fastboot mode, use:

```
fastboot devices
```

Once you managed to get into fastboot mode, you can try [above steps](#) to update the Android/Linux image.

If you happen to have a [P2 board](#), you should be able to reset the Snapdragon to the recovery image by starting up the Snapdragon while shorting the two pins next to where J3 is written (The two rectangular pins in-between the corner hole and the MicroSD card slot almost at the edge of the board).

If everything fails, you probably need to request help from intrinsyc.

No space left on device

Sometimes `make eagle_default upload` fails to upload:

```
failed to copy 'px4' to '/home/linaro/px4': No space left on device
```

This can happen if ramdumps fill up the disk. To clean up, do:

```
rm -rf /var/log/ramdump/*
```

Also, the logs might have filled the space. To delete them, do:

```
rm -rf /root/log/*
```

Undefined PLT symbol

_FDtest

If you see the following output on mini-dm when trying to start the px4 program, it means that you need to [update the ADSP firmware](#):

```
[08500/03] 05:10.960 HAP:45:undefined PLT symbol _FDtest (689) /libpx4muorb_skel.so 03
```

Something else

If you have changed the source, presumably added functions and you see `undefined PLT symbol ...` it means that the linking has failed.

- Do the declaration and definition of your function match one to one?
- Is your code actually getting compiled? Is the module listed in the [cmake config](#)?
- Is the (added) file included in the `CMakeLists.txt` ?
- Try adding it to the POSIX build and running the compilation. The POSIX linker will inform you about linking errors at compile/linking time.

krait update param XXX failed on startup

```
ERROR [platforms_posix_px4_layer] krait update param 297 failed
ERROR [platforms_posix_px4_layer] krait update param 646 failed

px4 starting.
ERROR [muorb] Initialize Error calling the uorb fastrpc initialize method..
ERROR [muorb] ERROR: FastRpcWrapper Not Initialized
```

If you get errors like the above when starting px4, try

- [upgrading the Linux image](#)
- and [updating the ADSP firmware](#). Also try to do this from a native Linux installation instead of a virtual machine. There have been [reports](#) where it didn't seem to work when done in a virtual machine.
- then [rebuild the px4 software](#), by first completely deleting your existing Firmware repo and then recloning it [as described here](#)
- and finally [rebuild and re-run it](#)
- make sure the executable bit of `/usr/local/qr-linux/q6-admin.sh` is set: `adb shell chmod +x /usr/local/qr-linux/q6-admin.sh`

ADSP restarts

If the mini-dm console suddenly shows a whole lot of INIT output, the ADSP side has crashed. The reasons for it are not obvious, e.g. it can be some segmentation fault, null pointer exception, etc..

The mini-dm console output typically looks like this:

```
[08500/02] 20:32.332 Process Sensor launched with ID=1 0130 main.c
[08500/02] 20:32.337 mmpm_register: MMPM client for USM ADSP core 12 0117 UltrasoundS
[08500/02] 20:32.338 ADSP License DB: License validation function with id 164678 stored
[08500/02] 20:32.338 AvsCoreSvc: StartSvcHandler Enter 0518 AdspCoreSvc.cpp
[08500/02] 20:32.338 AdspCoreSvc: Started successfully 0534 AdspCoreSvc.cpp
[08500/02] 20:32.342 DSPS INIT 0191 sns_init.dsps.c
[08500/02] 20:32.342 INIT DONE 0224 sns_init.dsps.c
[08500/02] 20:32.342 Sensors Init : waiting(1) 0160 sns_init.dsps.c
[08500/02] 20:32.342 INIT DONE 0224 sns_init.dsps.c
[08500/02] 20:32.342 THRD CREATE: Thread=0x39 Name(Hex)= 53, 4e, 53, 5f, 53, 4d, 47, 52
[08500/02] 20:32.343 THRD CREATE: Thread=0x38 Name(Hex)= 53, 4e, 53, 5f, 53, 41, 4d, 0
[08500/02] 20:32.343 THRD CREATE: Thread=0x37 Name(Hex)= 53, 4e, 53, 5f, 53, 43, 4d, 0
[08500/02] 20:32.343 THRD CREATE: Thread=0x35 Name(Hex)= 53, 4e, 53, 5f, 50, 4d, 0, 0
[08500/02] 20:32.343 THRD CREATE: Thread=0x34 Name(Hex)= 53, 4e, 53, 5f, 53, 53, 4d, 0
[08500/02] 20:32.343 THRD CREATE: Thread=0x33 Name(Hex)= 53, 4e, 53, 5f, 44, 45, 42, 55
[08500/02] 20:32.343 Sensors Init : ///////////init once completed////////// 0169 sn
[08500/02] 20:32.342 loading BLSP configuration 0189 blsp_config.c
[08500/02] 20:32.343 Sensors DIAG F3 Trace Buffer Initialized 0260 sns_init.dsps.c
[08500/02] 20:32.345 INIT DONE 0224 sns_init.dsps.c
[00053/03] 20:32.345 Unsupported algorithm service id 0 0953 sns_scm_ext.c
[08500/02] 20:32.346 INIT DONE 0224 sns_init.dsps.c
[08500/02] 20:32.347 INIT DONE 0224 sns_init.dsps.c
[08500/02] 20:32.347 INIT DONE 0224 sns_init.dsps.c
[08500/02] 20:32.546 HAP:159:unable to open the specified file path 0167 file.c
[08500/04] 20:32.546 failed to open /usr/share/data/adsp/blsp.config 0204 blsp_config
[08500/04] 20:32.546 QDSP6 Main.c: blsp_config_load() failed 0261 main.c
[08500/02] 20:32.546 Loaded default UART-BAM mapping 0035 blsp_config.c
[08500/02] 20:32.546 UART tty-1: BAM-9 0043 blsp_config.c
[08500/02] 20:32.546 UART tty-2: BAM-6 0043 blsp_config.c
[08500/02] 20:32.546 UART tty-3: BAM-8 0043 blsp_config.c
[08500/02] 20:32.546 UART tty-4: BAM-2 0043 blsp_config.c
[08500/02] 20:32.546 UART tty-5: BAM N/A 0048 blsp_config.c
[08500/02] 20:32.546 UART tty-6: BAM N/A 0048 blsp_config.c
[08500/02] 20:32.547 HAP:111:cannot find /oemconfig.so 0141 load.c
[08500/03] 20:32.547 HAP:4211::error: -1: 0 == dynconfig_init(&conf, "security") 0696
[08500/02] 20:32.548 HAP:76:cannot find /voiceproc_tx.so 0141 load.c
[08500/02] 20:32.550 HAP:76:cannot find /voiceproc_rx.so 0141 load.c
```

Do I have a P1 or P2 board?

The silkscreen on the Snapdragon reads something like:

1DN14-25-
H9550-P1
REV A
QUALCOMM

If you see **H9550**, it means you have a P2 board!

Please ignore that it says -P1.

Presumably P1 boards don't have a factory partition/image and therefore can't be restored to factory state.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22

12:56:03

获取I/O数据

官网英文原文地址：<http://dev.px4.io/advanced-accessing-io-data.html>

Low level bus data can be accessed from code running on the aDSP, using a POSIX-like API called DSPAL. The header files for this API are maintained on [github](#) and are commented with Doxygen formatted documentation in each header file. A description of the API's supported and links to the applicable header files is provided below.

API Overview

- Serial: [Serial.h](#)
- I2C: [I2C.h](#)
- SPI: [SPI.h](#)
- GPIO: [GPIO.h](#)
- Timers: [qurt_timer.h](#)
- Power Control: [HAP_power.h](#)

Sample Source Code

The unit test code to verify each DSPAL function also represent good examples for how to call the functions.

This code is also on [github](#)

Setting the Serial Data Rate

The serial API does not conform to the termios convention for setting data rate through the tcsetattr() function. IOCTL codes are used instead and are described in the header file linked above.

Timers

Additional functions for more advanced aDSP operations are available with the prefix qurt_. Timer functions, for example, are available with the qurt_timer prefix and are documented in the qurt_timer.h header file included with the [Hexagon SDK](#).

Setting the Power Level

Using the HAP functions provided by the Hexagon SDK, it is possible to set the power level of the aDSP. This will often lead to reduced I/O latencies. More information on these API's is available in the [HAP_power.h](#) header file available in the [Hexagon SDK](#).

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Snapdragon: Camera and Optical Flow

官网英文原文地址：http://dev.px4.io/advanced-snapdragon_camera.html

Please follow the following instructions to use the camera and optical flow with a Snapdragon Flight.

Snapdragon: Camera driver

This [package](#) provides tools to work with the Snapdragon Flight cameras as well as perform optical flow for use with the PX4 flight stack. The package can be used with ROS by building with catkin or, alternatively, with pure cmake, where only the executables that do not depend on ROS will be built.

The package is to be compiled on the Snapdragon board. Two variants are provided: Building with ROS, where all features are available, and building with pure CMake, where only ROS-independent applications are compiled (including the optical flow node).

Building with pure CMake

For the pure CMake install variant, clone the required repositories in a directory, e.g. `~/src` :

```
cd ~/src
git clone https://github.com/ethz-ait/klt_feature_tracker.git
git clone https://github.com/PX4/snap_cam.git
```

Initialize the Mavlink submodule:

```
cd snap_cam
git submodule init
git submodule update --recursive
```

Compile with:

```
mkdir -p build
cd build
cmake ..
make
```

Run the optical flow application with (note that you need to be root for this):

```
./optical_flow [arguments ...]
```

Building with ROS

Prerequisites

To run the ROS nodes on the Snapdragon Flight, ROS indigo has to be installed. Follow [this](#) link to install it on your Snapdragon Flight. (preferably using the linaro user: `$ su linaro`)

If you're having permission issues while installing ros try

```
sudo chown -R linaro:linaro /home/linaro
```

Install the following dependencies:

ROS dependencies

```
sudo apt-get install ros-indigo-mavlink ros-indigo-tf ros-indigo-orocos-toolchain ros-ind
```

Others

```
sudo apt-get install libeigen3-dev sip-dev libyaml-cpp-dev
```

To install OpenCV, [download](#) and push the `.deb` package to the Snapdragon and install it using

```
adb push /path/to/file /home/linaro/  
dpkg -i opencv3_20160222-1_armhf.deb
```

or use

```
sudo apt-get install ros-indigo-opencv3
```

create a catkin workspace

Next, create a catkin workspace (e.g. in `/home/linaro`)

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ..
catkin_make
```

Then clone the following four catkin packages and build

```
cd src
git clone https://github.com/ros-perception/vision_opencv
git clone https://github.com/ros-perception/image_common
git clone https://github.com/ethz-ait/klt_feature_tracker.git
git clone https://github.com/PX4/snap_cam.git
cd ..
catkin_make
```

Image publisher node

Once your catkin workspace is built and sourced you can start the image publisher using

```
roslaunch snap_cam <CAM>.launch
```

where `<CAM>` is either `optflow` or `highres` to stream the optical flow or high resolution cameras, respectively. You can set the parameters (camera, resolution and fps) in the launch files (`pathToYourCatkinWs/src/snap_cam/launch/<cam>.launch`)

You can now subscribe to the images in your own ROS node.

Camera calibration

For optical flow computations, a calibration file needs to be used. This package contains default calibration files for VGA and QVGA resolution. Nevertheless, we recommend calibrating your camera (see below) for better performance. For this you must build this package with catkin as described above and launch the optical flow image publisher:

```
roslaunch snap_cam optflow.launch
```

Clone and build this package in a catkin workspace on your computer. Add any missing dependencies:

```
sudo apt-get install python-pyside
```

On your computer launch the calibration app:

```
export ROS_MASTER_URI=http://<snapdragon IP>:11311
roslaunch snap_cam cameraCalibrator.launch
```

NOTE: If your image topics are empty, make sure to set the environment variable ROS_IP to the respective IP on both devices.

Set the appropriate checkerboard parameters in the app. Start recording by clicking on the button and record your checkerboard from sufficiently varying angles. Once done, click stop recording. The camera calibration will be written to

`pathToYourCatkinWs/src/snap_cam/calib/cameraParameters.yaml`. Push this file to your snapdragon.

```
adb push /pathToYourCatkinWs/src/snap_cam/calib/cameraParameters.yaml pathToSnapCam/calib
```

Running the optical flow

With pure CMake

Run the following in your build directory:

```
./optical_flow [arguments ...]
```

All arguments are optional.

- `-r` specifies the camera resolution. The default is `VGA`. Valid resolutions are `VGA` and `QVGA`.
- `-f` specifies the camera frame-rate. The default is 15. Valid values are 15, 24, 30, 60, 90, 120.
- `-n` specifies the number of features with which to compute the optical flow. The default is 10.
- `-c` specifies the calibration file. The default is `../calib/<resolution>/cameraParameters.yaml`.

With ROS

After sourcing your workspace with `source ~/catkin_ws/devel/setup.bash`, run:

```
rosrun snap_cam optical_flow [arguments ...]
```

The arguments are the same as for the pure CMake build/=.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

中间件及架构

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

uORB消息机制

官网英文原文地址：<http://dev.px4.io/advanced-uorb.html>

简介

uORB是一种用于进程间进行异步发布和订阅的消息机制API。

在[教程](#)中可以学习通过C++如何使用uORB。

由于很多应用都是基于uORB的，因此在系统刚启动时uORB就自动运行了。uORB通过 `uorb start` 启动。可以使用 `uorb test` 进行单位测试。

添加新的主题(topic)

要想增加新的topic，你需要在 `msg/` 目录下创建一个新的 `.msg` 文件并在 `msg/CMakeLists.txt` 下添加该文件。这样C/C++编译器自动在程序中添相应的代码。

可以先看看现有的 `msg` 文件了解下都支持那些类型。一个消息也可以嵌套在其他消息当中。

每一个生成的C/C++结构体中，会多出一个 `uint64_t timestamp` 字段。这个变量用于将消息记录到日志当中。

为了在代码中使用"topic"需要添加头文件:

```
#include <uORB/topics/topic_name.h>
```

首先需要在文件 `.msg` 中，通过添加类似如下的一行代码，一个消息定义就可以用于多个独立的主题。

```
# TOPICS mission offboard_mission onboard_mission
```

【按】这里这一步将产生三个主题ID- `mission`、`offboard_mission`、以及 `onboard_mission` (第一个ID务必与.msg文件名相同)

然后在代码中，把它们作为主题ID用: `ORB_ID(offboard_mission)` .

发布主题

在系统的任何地方都可以发布一个主题，包括在中断上下文中(被 `hrt_call` 接口调用的函数).但是，公告(advertise)一个主题仅限于在中断上下文之外.

一个主题只能由同一个进程进行公告，并作为其之后的发布(publish).

列出所有主题并进行监听

`接收者(listener)` 命令仅在 Pixracer (FMUv4) 以及 Linux/OS X 上可用。

要列出所有主题，先列出文件句柄：

```
ls /obj
```

要列出一个主题中的5个消息，执行以下监听命令：

```
listener sensor_accel 5
```

得到的输出就是关于该主题的n次内容：

```
TOPIC: sensor_accel #3
timestamp: 84978861
integral_dt: 4044
error_count: 0
x: -1
y: 2
z: 100
x_integral: -0
y_integral: 0
z_integral: 0
temperature: 46
range_m_s2: 78
scaling: 0

TOPIC: sensor_accel #4
timestamp: 85010833
integral_dt: 3980
error_count: 0
x: -1
y: 2
z: 100
x_integral: -0
y_integral: 0
z_integral: 0
temperature: 46
range_m_s2: 78
scaling: 0
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

创建自定义MAVLink消息

官网英文原文地址：<http://dev.px4.io/custom-mavlink-message.html>

这篇教程是假设你已经在 `msg/ca_trajectory.msg` 有了一个自定义uORB `ca_trajectory` 消息，并且在 `mavlink/include/mavlink/v1.0/custom_messages/mavlink_msg_ca_trajectory.h` 有了一个自定义mavlink `ca_trajectory` 消息（点击[此处](#)查看如何建立一个自定义mavlink消息以及头文件）。

发送自定义MAVLink消息

这部分介绍如何使用一个自定义uORB消息并且作为mavlink消息发送。添加 `mavlink` 的头文件和uorb消息到 `mavlink_messages.cpp`

```
#include <uORB/topics/ca_trajectory.h>
#include <v1.0/custom_messages/mavlink_msg_ca_trajectory.h>
```

在 `mavlink_messages.cpp` 中创建一个新的类

```
class MavlinkStreamCaTrajectory : public MavlinkStream
{
public:
    const char *get_name() const
    {
        return MavlinkStreamCaTrajectory::get_name_static();
    }
    static const char *get_name_static()
    {
        return "CA_TRAJECTORY";
    }
    uint8_t get_id()
    {
        return MAVLINK_MSG_ID_CA_TRAJECTORY;
    }
    static MavlinkStream *new_instance(Mavlink *mavlink)
    {
        return new MavlinkStreamCaTrajectory(mavlink);
    }
    unsigned get_size()
    {
        return MAVLINK_MSG_ID_CA_TRAJECTORY_LEN + MAVLINK_NUM_NON_PAYLOAD_BYTES;
    }
}
```

```

private:
    MavlinkOrbSubscription *_sub;
    uint64_t _ca_traj_time;

    /* do not allow top copying this class */
    MavlinkStreamCaTrajectory(MavlinkStreamCaTrajectory &);
    MavlinkStreamCaTrajectory& operator = (const MavlinkStreamCaTrajectory &);

protected:
    explicit MavlinkStreamCaTrajectory(Mavlink *mavlink) : MavlinkStream(mavlink),
        _sub(_mavlink->add_orb_subscription(ORB_ID(ca_trajectory))), // make sure you en
        _ca_traj_time(0)
    {}

    void send(const hrt_abstime t)
    {
        struct ca_traj_struct_s _ca_trajectory; //make sure ca_traj_struct_s is the de

        if (_sub->update(&_ca_traj_time, &_ca_trajectory)) {
            mavlink_ca_trajectory_t _msg_ca_trajectory; //make sure mavlink_ca_trajectory

            _msg_ca_trajectory.timestamp = _ca_trajectory.timestamp;
            _msg_ca_trajectory.time_start_usec = _ca_trajectory.time_start_usec;
            _msg_ca_trajectory.time_stop_usec = _ca_trajectory.time_stop_usec;
            _msg_ca_trajectory.coefficients = _ca_trajectory.coefficients;
            _msg_ca_trajectory.seq_id = _ca_trajectory.seq_id;

            _mavlink->send_message(MAVLINK_MSG_ID_CA_TRAJECTORY, &_msg_ca_trajectory);
        }
    }
};


```

最后附加流类 `streams_list` 的到[mavlink_messages.cpp](#)底部

```

StreamListItem *streams_list[] = {
    ...
new StreamListItem(&MavlinkStreamCaTrajectory::new_instance, &MavlinkStreamCaTrajectory::
nullptr
};


```

接收自定义MAVLink消息

这部分解释如何通过mavlink接收消息并将其发布到uORB。

在[mavlink_receiver.h](#)中增加一个用来处理接收信息得函数

```
#include <uORB/topics/ca_trajectory.h>
#include <v1.0/custom_messages/mavlink_msg_ca_trajectory.h>
```

在 `mavlink_receiver.h` 中增加一个处理类 `MavlinkReceiver` 中的输入 `mavlink` 消息的函数

```
void handle_message_ca_trajectory_msg(mavlink_message_t *msg);
```

在 `mavlink_receiver.h` 中加入一个类 `MavlinkReceiver` 中的 uORB 消息发布者

```
orb_advert_t _ca_traj_msg_pub;
```

在 `mavlink_receiver.cpp` 中实现 `handle_message_ca_trajectory_msg` 功能

```
void
MavlinkReceiver::handle_message_ca_trajectory_msg(mavlink_message_t *msg)
{
    mavlink_ca_trajectory_t traj;
    mavlink_msg_ca_trajectory_decode(msg, &traj);

    struct ca_traj_struct_s f;
    memset(&f, 0, sizeof(f));

    f.timestamp = hrt_absolute_time();
    f.seq_id = traj.seq_id;
    f.time_start_usec = traj.time_start_usec;
    f.time_stop_usec = traj.time_stop_usec;
    for(int i=0;i<28;i++)
        f.coefficients[i] = traj.coefficients[i];

    if (_ca_traj_msg_pub == nullptr) {
        _ca_traj_msg_pub = orb_advertise(ORB_ID(ca_trajectory), &f);

    } else {
        orb_publish(ORB_ID(ca_trajectory), _ca_traj_msg_pub, &f);
    }
}
```

最后确定函数在 `MavlinkReceiver::handle_message()` 中被调用

```
MavlinkReceiver::handle_message(mavlink_message_t *msg)
{
    switch (msg->msgid) {
        ...
        case MAVLINK_MSG_ID_CA_TRAJECTORY:
            handle_message_ca_trajectory_msg(msg);
            break;
        ...
    }
}
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

守护进程

官网英文原文地址：<http://dev.px4.io/architecture-daemon.html>

守护进程 `daemon` 是运行在后台的进程。在 NuttX 中守护进程是一个任务，在 POSIX(Linux/Mac OS) 中是一个线程

```
daemon_task = px4_task_spawn_cmd("commander",
                                  SCHED_DEFAULT,
                                  SCHED_PRIORITY_DEFAULT + 40,
                                  3600,
                                  commander_thread_main,
                                  (char * const *)&argv[0]);
```

以下是参数：

- `arg0`: 进程名 `commander`
- `arg1`: 调度类型 (RR or FIFO) the scheduling type (RR or FIFO)
- `arg2`: 调度优先级
- `arg3`: 新进程或线程堆栈大小
- `arg4`: 任务/线程主函数
- `arg5`: 一个void指针传递给新任务,在这种情况下是命令行参数

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

驱动框架

官网英文原文地址：<http://dev.px4.io/advanced-drivers.html>

PX4的代码库使用一个轻量级的，统一的驱动抽象层：DriverFramework. POSIX和QuRT的驱动写入这个驱动框架当中。

旧的NuttX驱动是基于[设备](#)架构的，以后将会移植到驱动框架之中。

核心架构

PX4是一个反应式系统[reactive system](#)，使用订阅/发布来传递消息。文件句柄是不被操作系统的核心所需要或者使用。主要使用了以下两个API：

- 发布/订阅系统，该系统拥有一个文件，网络或者共享内存，其依靠于PX4后台运行。
- 全局驱动注册器，它允许枚举设备和获取/设置这些设备参数。这个可以很简单的作为一个链表或者文件系统地图。

一个新的平台

NuttX

- 启动脚本位于[ROMFS/px4fmu_common](#)
- 系统配置文件位于[nuttx-configs](#)。作为应用的一部分被构建并被操作系统加载。
- PX4中间件配置位于[src/drivers/boards](#).其中包括总线和GPIO映射还有硬件平台初始化代码。
- 驱动位于[src/drivers](#)
- 参考配置：运行使px4fmu-v4_default构建FMUv4配置，这是当前NuttX参考配置。

QuRT / Hexagon

- 启动脚本位于[posix-configs/](#)
- 系统配置文件模式作为Linux映射的一部分（备注：提供本地的LINUX IMAGE和flash指令）
- PX4中间件配置位于[src/drivers/boards](#)。备注：增加总线配置。
- 驱动位于[DriverFramework](#)
- 参考配置：运行'make qurt_eagle_release'构建Snapdragon飞行参考配置。

驱动ID

PX4使用驱动ID用于将独立传感器贯穿于整个系统。这些ID被存储于配置参数，被用于匹配传感器校正值，以及决定哪些传感器被记录到log中。

传感器的顺序（例如一个是 /dev/mag0，另一个是 /dev/mag1）于优先级不挂钩的，优先级实际是在发布uORB topic时确定的。

举个例子

用三个磁力计作为例子，使用飞行log (.px4log) 转存变量。磁力计被作为主要传感器，而三个参数编码为传感器ID和 MAG_PRIME。每一个MAGx_ID是一个24bit数值，左面手工填零补充。

```
CAL_MAG0_ID = 73225.0
CAL_MAG1_ID = 66826.0
CAL_MAG2_ID = 263178.0
CAL_MAG_PRIME = 73225.0
```

通过I2C连接的外部HMC5983，总线1，地址0x1E：在log中以 IMU.MagX 显示。

```
# device ID 73225 in 24-bit binary:
00000001 00011110 00001 001

# decodes to:
HMC5883 0x1E bus 1 I2C
```

通过SPI连接的内部HMC5983，总线1，选择slot5。在log中以 IMU1.MagX 显示。

```
# device ID 66826 in 24-bit binary:
00000001 00000101 00001 010

# decodes to:
HMC5883 dev 5 bus 1 SPI
```

以及通过SPI总线连接的内部MPU9250磁力计，总线1，从设备选择slot4。在log中以 IMU2.MagX 显示。

```
# device ID 263178 in 24-bit binary:
00000100 00000100 00001 010

#decodes to:
MPU9250 dev 4 bus 1 SPI
```

设备ID编码

根据此格式，设备ID是一个24位的数字。注意，第一字段是上述解码示例中的最低有效位。

```
struct DeviceStructure {
    enum DeviceBusType bus_type : 3;
    uint8_t bus: 5;      // which instance of the bus type
    uint8_t address;    // address on the bus (eg. I2C address)
    uint8_t devtype;    // device class specific device type
};
```

这里 `bus_type` 按以下方式解码：

```
enum DeviceBusType {
    DeviceBusType_UNKNOWN = 0,
    DeviceBusType_I2C     = 1,
    DeviceBusType_SPI     = 2,
    DeviceBusType_UAVCAN  = 3,
};
```

`devtype` 按以下方式解码：

```
#define DRV_MAG_DEVTYPE_HMC5883 0x01
#define DRV_MAG_DEVTYPE_LSM303D 0x02
#define DRV_MAG_DEVTYPE_ACCELSIM 0x03
#define DRV_MAG_DEVTYPE_MPU9250 0x04
#define DRV_ACC_DEVTYPE_LSM303D 0x11
#define DRV_ACC_DEVTYPE_BMA180 0x12
#define DRV_ACC_DEVTYPE_MPU6000 0x13
#define DRV_ACC_DEVTYPE_ACCELSIM 0x14
#define DRV_ACC_DEVTYPE_GYROSIM 0x15
#define DRV_ACC_DEVTYPE_MPU9250 0x16
#define DRV_GYR_DEVTYPE_MPU6000 0x21
#define DRV_GYR_DEVTYPE_L3GD20 0x22
#define DRV_GYR_DEVTYPE_GYROSIM 0x23
#define DRV_GYR_DEVTYPE_MPU9250 0x24
#define DRV_RNG_DEVTYPE_MB12XX 0x31
#define DRV_RNG_DEVTYPE_LL40LS 0x32
```

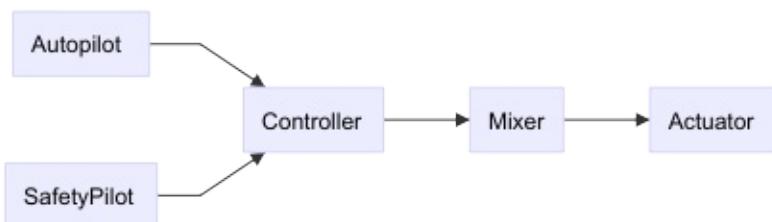
机型

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

机型综述

官网英文原文地址：<http://dev.px4.io/airframes-architecture.html>

PX4系统是模块化的架构，这使得它对所有的机器人类型都可以使用同一个代码库。



基本设备

在机型部分所用到的硬件包括以下基本设备：

- 1个Taranis Plus遥控器（或者其它有PPM/S.BUS输出的设备），用于保证安全飞行。
- 1个地面站
 - Samsung Note 4或者同类型的较新的Android平板
 - iPad（需要无线遥测适配器）
 - 任何MacBook或者Ubuntu Linux笔记本
- 1台in-field电脑（用于软件开发者）
 - MacBook Pro或者Air，至少OS X 10.10
 - 现代Ubuntu Linux笔记本，至少14.04
- 安全眼镜
 - 用于多旋翼危险测试

PX4应用范围很广，但是对于新手开发者而言，从标准配置做起会更好，1个Taranis Plus遥控器，1个Note 4平板就可以组成一套便宜的套件。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

添加一个新的机型

官网英文原文地址：<http://dev.px4.io/airframes-adding-a-new-frame.html>

PX4使用存储的配置作为机型的起始点。添加配置是非常简单的：在[init.d文件夹](#)创建一个新的文件，这个文件需要以一个没有使用的自动启动ID作为文件名的前缀，然后[构建并上传固件](#)即可。

如果不想创建自己的配置文件，也可以用SD卡上的文本文件替换掉已有的自定义配置文件，具体细节请查看[自定义系统启动页](#)。

机型配置

一个机型配置包括3项基本内容：

- 应该启动的应用，例如多旋翼或者固定翼的控制器
- 系统（固定翼，飞翼或者多旋翼）的物理配置，这叫做混控器
- 参数整定

这三方面大多数时候是独立的，也就是说，许多配置会共享相同的机型物理布局以及启动相同的应用，它们之间最大的不同在参数整定部分。

所有的配置存储在[ROMFS/px4fmu_common/init.d](#)文件夹。所有的混控器存储在[ROMFS/px4fmu_common/mixers](#)文件夹。

配置文件

如下所示，是一个典型的配置文件：

```

#!/bin/sh

#
# @name Wing Wing (aka Z-84) Flying Wing
#
# @url https://pixhawk.org/platforms/planes/z-84_wing_wing
#
# @type Flying Wing
#
# @output MAIN1 left aileron
# @output MAIN2 right aileron
# @output MAIN4 throttle
#
# @output AUX1 feed-through of RC AUX1 channel
# @output AUX2 feed-through of RC AUX2 channel
# @output AUX3 feed-through of RC AUX3 channel
#
# @maintainer Lorenz Meier <lorenz@px4.io>
#

sh /etc/init.d/rc.fw_defaults

if [ $AUTOCNF == yes ]
then
    param set BAT_N_CELLS 2
    param set FW_AIRSPD_MAX 15
    param set FW_AIRSPD_MIN 10
    param set FW_AIRSPD_TRIM 13
    param set FW_ATT_TC 0.3
    param set FW_L1_DAMPING 0.74
    param set FW_L1_PERIOD 16
    param set FW_LND_ANG 15
    param set FW_LND_FLALT 5
    param set FW_LND_HHDIST 15
    param set FW_LND_HVIRT 13
    param set FW_LND_TLALT 5
    param set FW_THR_LND_MAX 0
    param set FW_PR_FF 0.35
    param set FW_RR_FF 0.6
    param set FW_RR_P 0.04
fi

# Configure this as plane
set MAV_TYPE 1
# Set mixer
set MIXER wingwing
# Provide ESC a constant 1000 us pulse
set PWM_OUT 4
set PWM_DISARMED 1000

```

注意事项: If you want to reverse a channel, never do this neither on your RC transmitter nor with e.g. `RC1_REV`. The channels are only reversed when flying in manual mode, when you switch in an autopilot flight mode, the channels output will still be wrong (it only inverts your RC signal). Thus for a correct channal assignment change either your PWM signals with `PWM_MAIN_REV1` (e.g. for channel one) or change the signs for both output scaling and output range in the corresponding mixer (see below).

混控器文件

一个典型的混控器文件会像下面这样：

舵机/电机的接口顺序和这个文件中的混控器顺序一致。

所以MAIN1对应左副翼，MAIN2对应右副翼，MAIN3置空（注意：Z即为空混控器），MAIN4则对应油门（对于一般固定翼配置，保持油门和输出4对应）。

混控器被编码为从-10000到10000的标准单位，对应-1到+1。

```
M: 2
O:      10000  10000      0 -10000  10000
S: 0 0   -6000  -6000      0 -10000  10000
S: 0 1    6500   6500      0 -10000  10000
```

从左到右每个数字代表的意思如下：

- M：代表有2个缩放系数（对应着两个输入）
- O：代表输出缩放系数（负输入量缩放系数为1，正输入量缩放系数为1），偏移量（这里是0），输出范围（这里-1到+1）
- S：代表第一个输入量的缩放系数：输入量来自控制组#0（姿态控制）的第一个输入（滚转），缩放系数为0.6，并且符号取反（-0.6换算到标准单位是-6000），没有偏移量（0），输出为全范围（-1到+1）
- S：代表第二个输入量的缩放系数：输入量来自控制组#0（姿态控制）的第二个输入（俯仰），缩放系数为0.65（0.65换算到标准单位是6500），没有偏移量（0），输出为全范围（-1到+1）

所有的缩放器结果累加，对飞翼而言，控制面偏移量取滚转信号的60%和俯仰信号的65%。如果俯仰信号和滚转信号都取最大值，那么偏移量将达到125%，超出了输出范围，这就意味着第一个通道（滚转）比第二个通道（俯仰）优先级高。

完整的混控器定义如下：

```
Delta-wing mixer for PX4FMU
=====
```

Designed for Wing Wing Z-84

This file defines mixers suitable for controlling a delta wing aircraft using PX4FMU. The configuration assumes the elevon servos are connected to PX4FMU servo outputs 0 and 1 and the motor speed control to output 3. Output 2 is assumed to be unused.

Inputs to the mixer come from channel group 0 (vehicle attitude), channels 0 (roll), 1 (pitch) and 3 (thrust).

See the README for more information on the scaler format.

Elevon mixers

```
Three scalers total (output, roll, pitch).
```

On the assumption that the two elevon servos are physically reversed, the pitch input is inverted between the two servos.

The scaling factor for roll inputs is adjusted to implement differential travel for the elevons.

M: 2

```
O:      10000  10000      0 -10000  10000
S: 0 0  -6000  -6000      0 -10000  10000
S: 0 1   6500   6500      0 -10000  10000
```

M: 2

```
O:      10000  10000      0 -10000  10000
S: 0 0  -6000  -6000      0 -10000  10000
S: 0 1  -6500  -6500      0 -10000  10000
```

Output 2

```
-----
This mixer is empty.
```

Z:

Motor speed mixer

```
-----
Two scalers total (output, thrust).
```

This mixer generates a full-range output (-1 to 1) from an input in the (0 - 1) range. Inputs below zero are treated as zero.

M: 1

```
O:      10000  10000      0 -10000  10000
S: 0 3     0  20000 -10000 -10000  10000
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

多旋翼

官网英文原文地址：<http://dev.px4.io/airframes-multicopter.html>

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

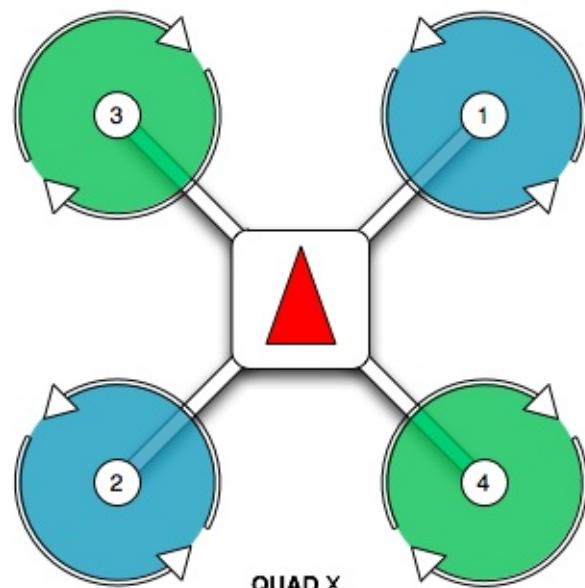
机型电机映射

官网英文原文地址：<http://dev.px4.io/airframes-motor-map.html>

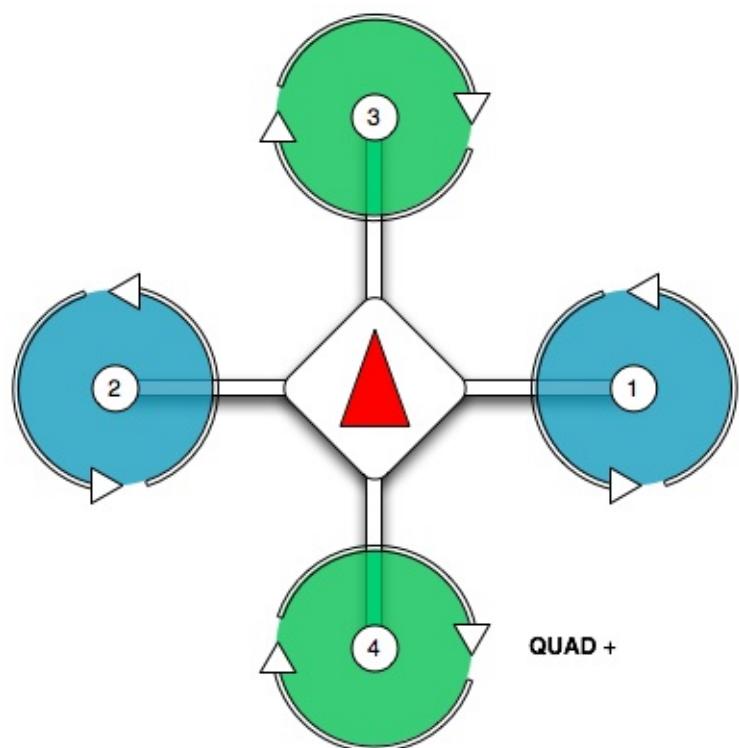
下面的电机映射指明'MAIN'输出口和电机的连接关系，同时也给出了螺旋桨的旋转方向。

交换三相电机三根连接线的其中两根，就可以改变电机的旋转方向。

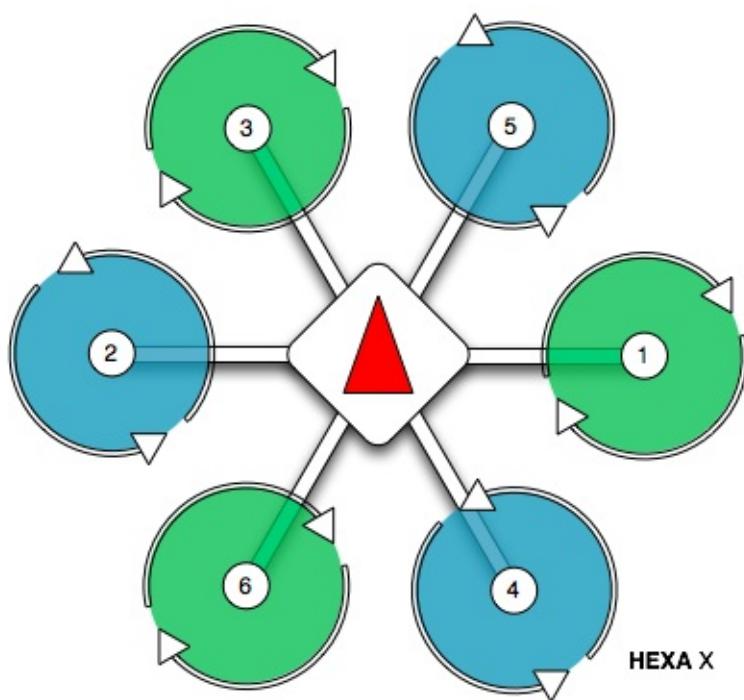
四旋翼X型布局



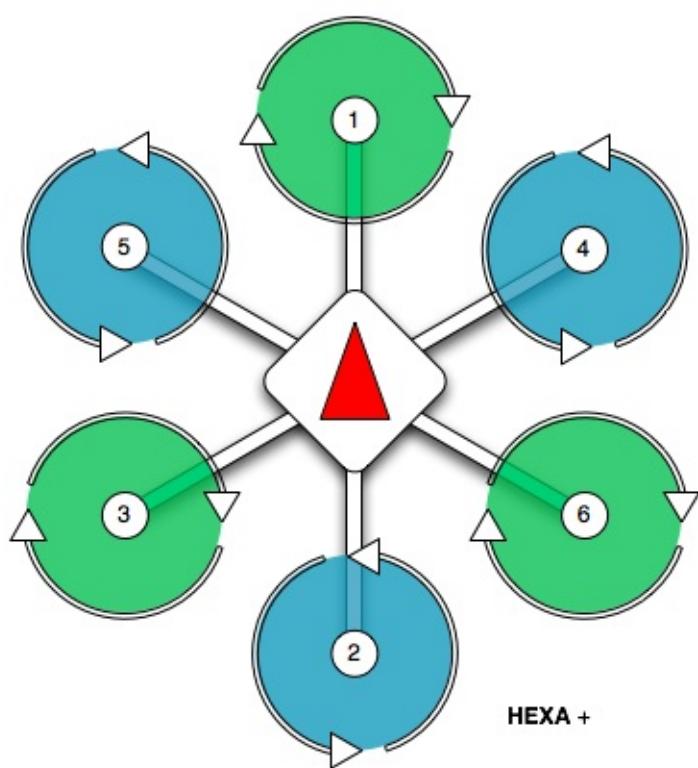
四旋翼+型布局



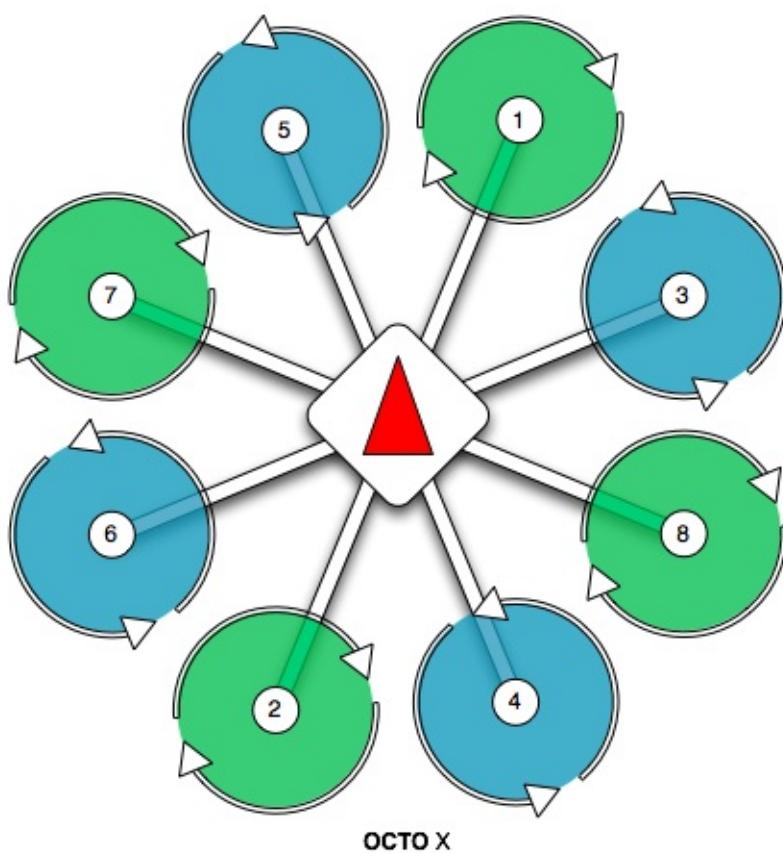
六旋翼X型布局



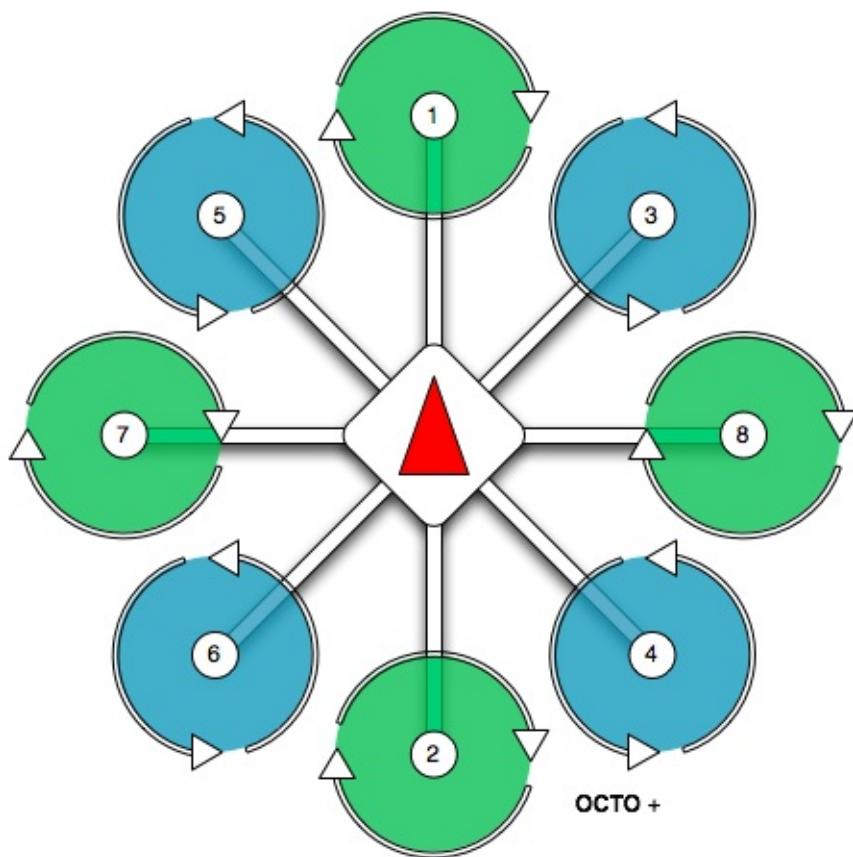
六旋翼+型布局



八旋翼X型布局



八旋翼+型布局



支持三旋翼和共轴六旋翼，但是目前没有布局图

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

QAV 250

官网英文原文地址：<http://dev.px4.io/airframes-multicopter-qav250.html>

部件列表

- Pixracer组件或者Pixfalcon组件 (包括GPS和电源模块)
- Mini telemetry set for HKPilot32

电机连接

输出	频率	作动器
MAIN1	400 Hz	右前方，逆时针
MAIN2	400 Hz	左后方，逆时针
MAIN3	400 Hz	左前方，顺时针
MAIN4	400 Hz	右后方，顺时针
AUX1	50 Hz	RC AUX1
AUX2	50 Hz	RC AUX2
AUX3	50 Hz	RC AUX3

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

Matrice 100

官网英文原文地址：<http://dev.px4.io/airframes-multicopter-matrice100.html>

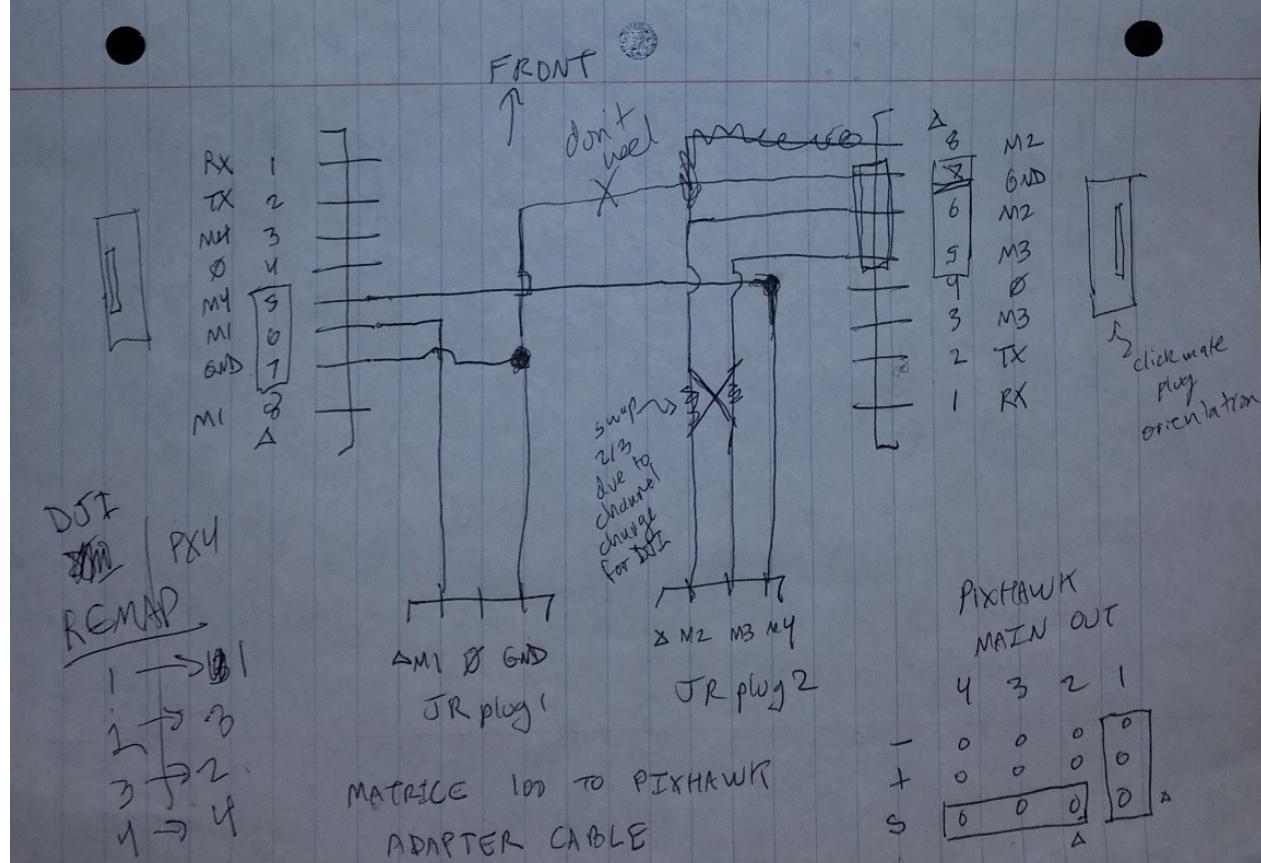
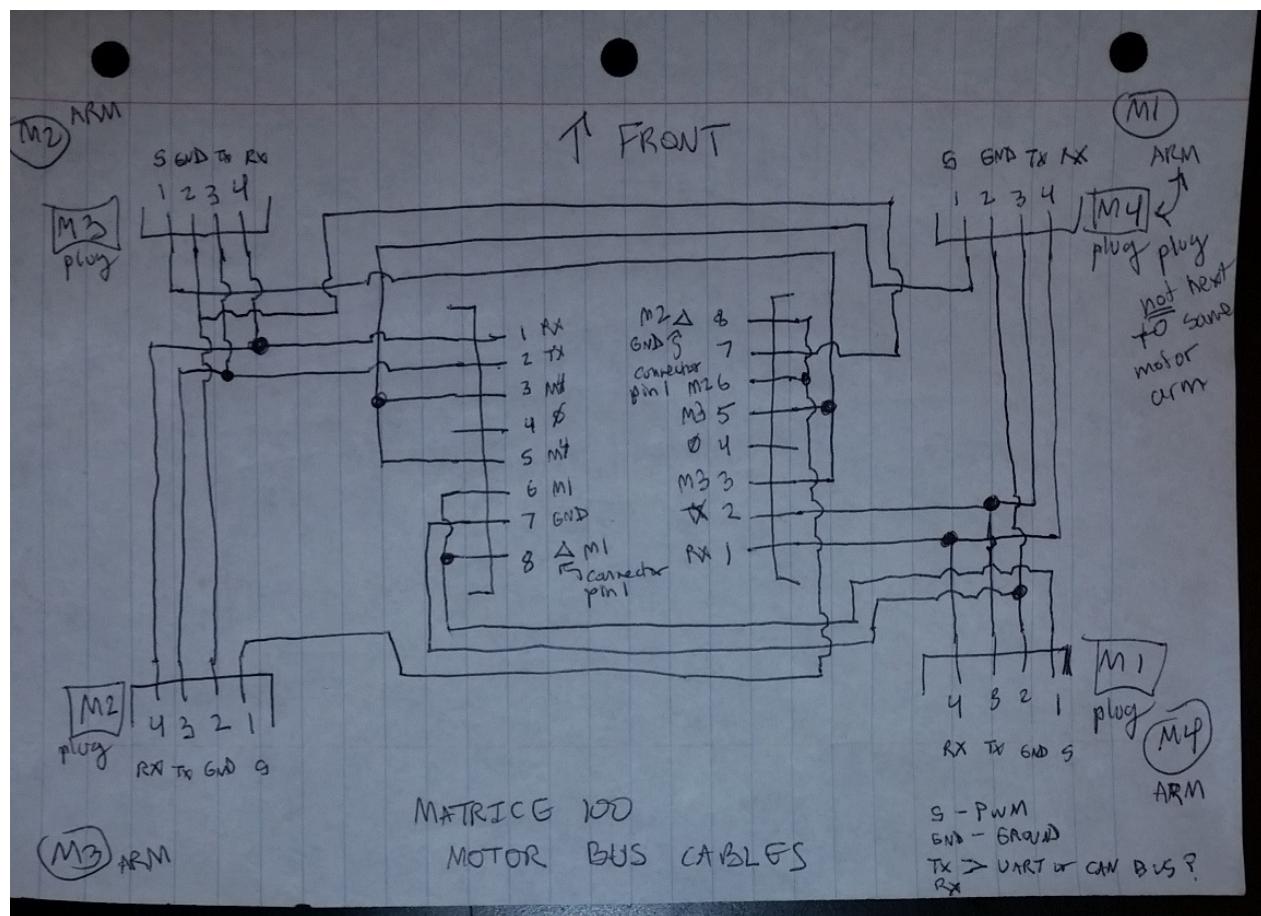
To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

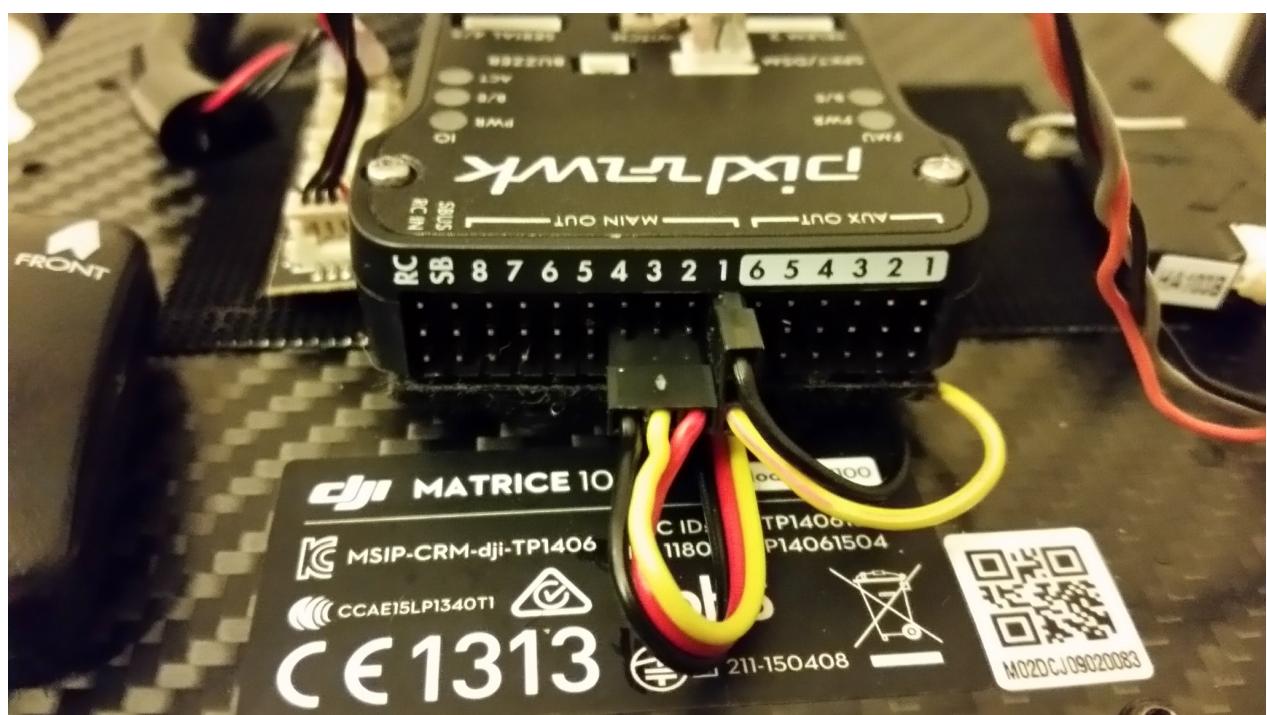
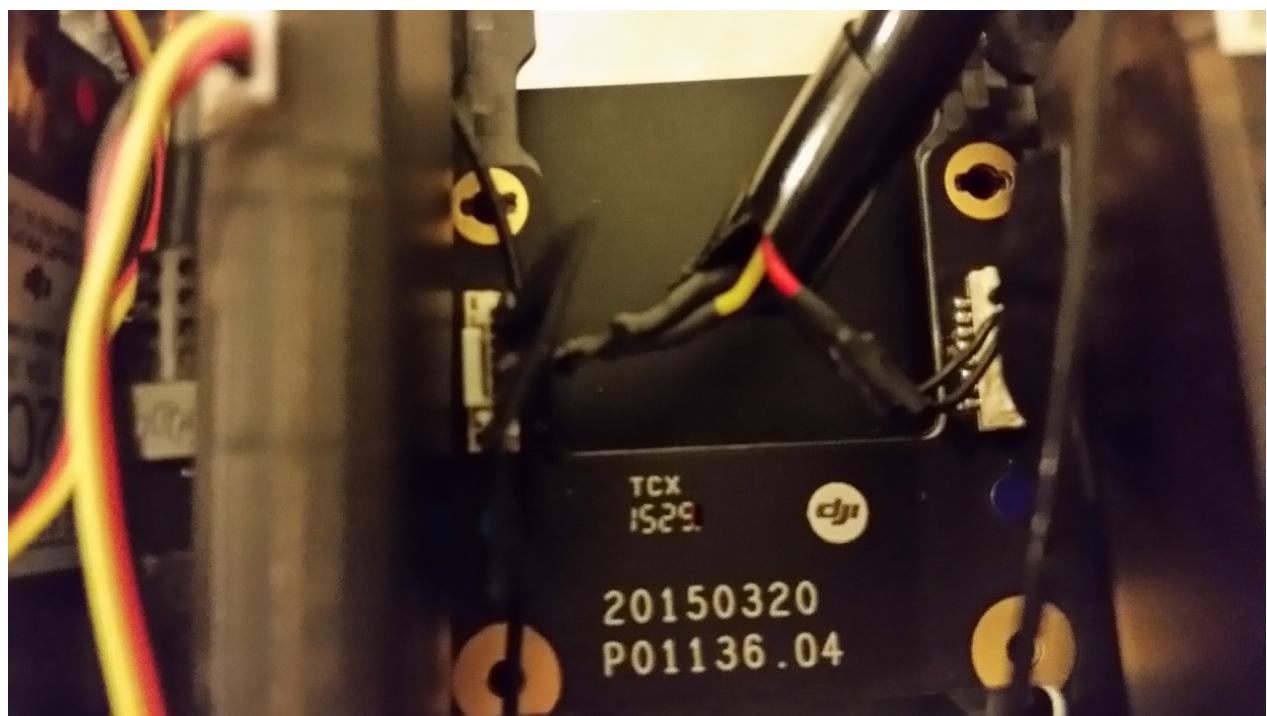


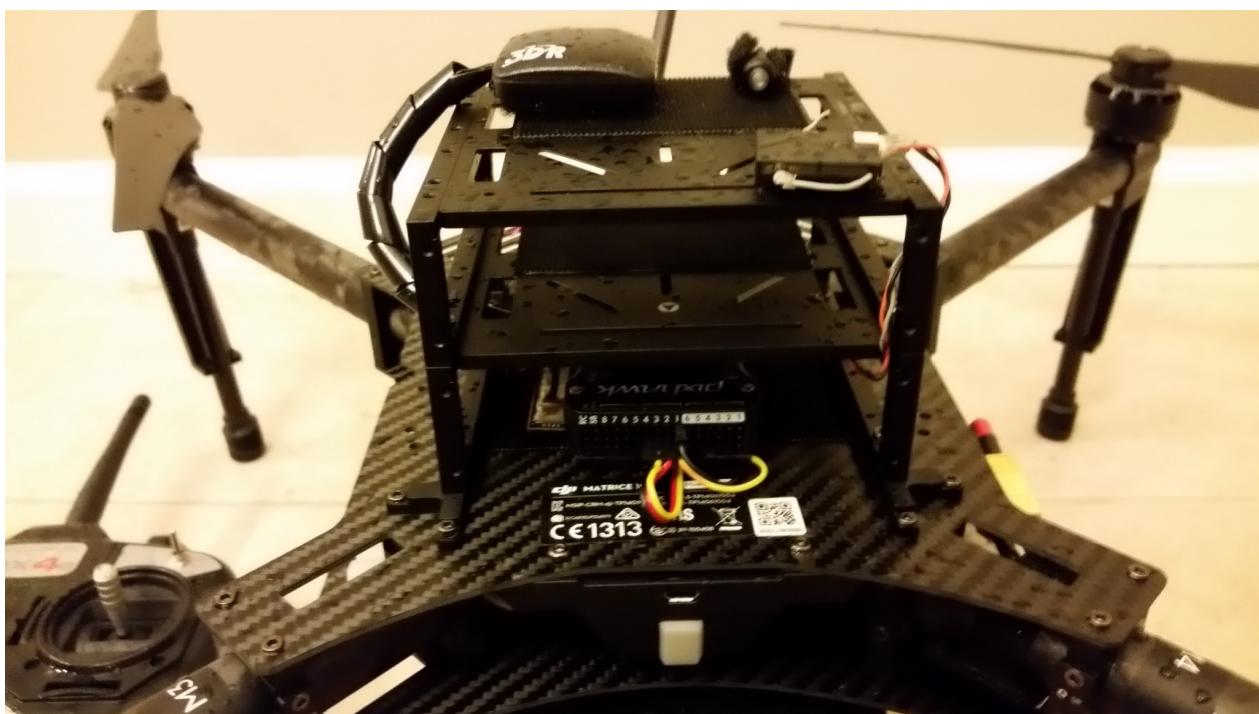
部件列表

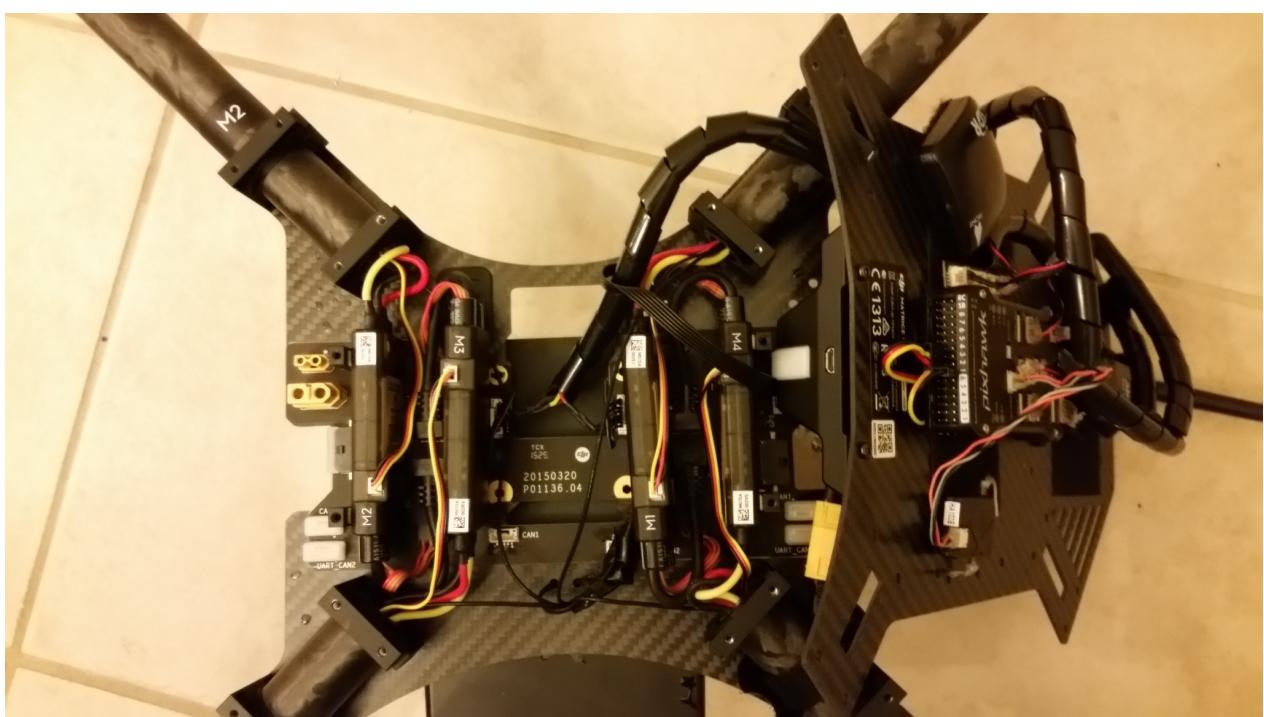
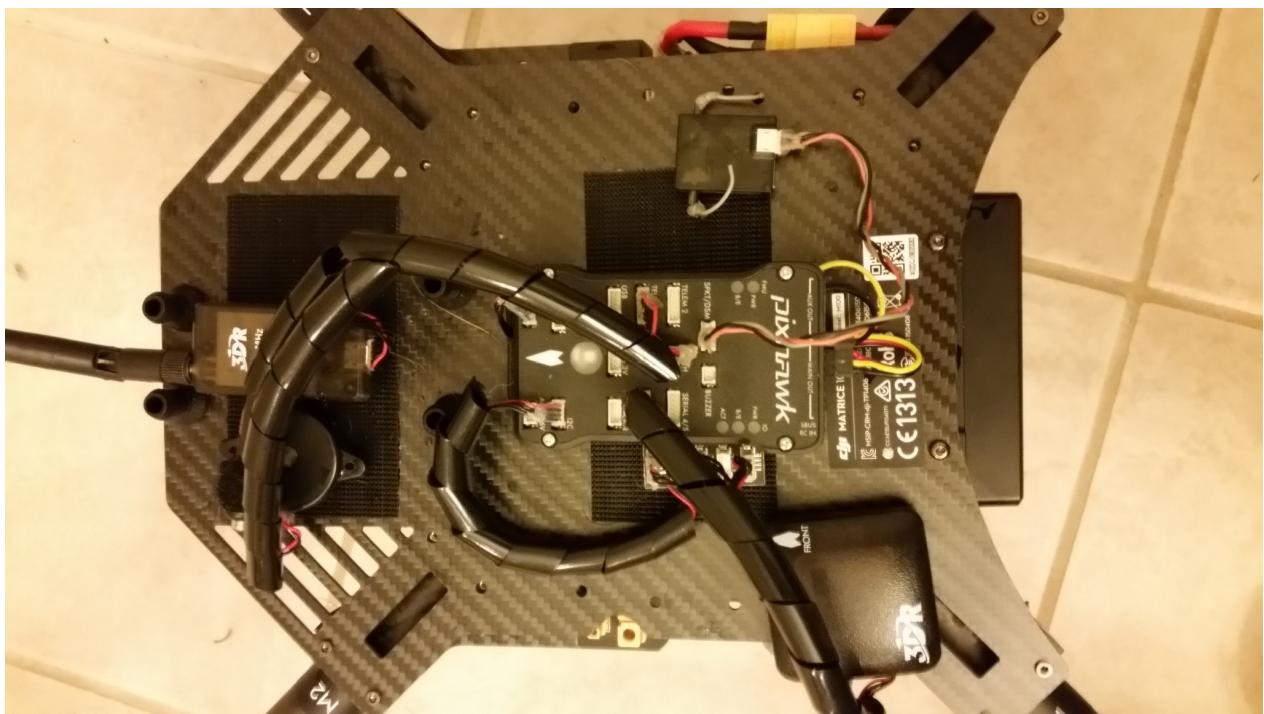
- DJI Matrice 100 仅包含电调，电机和机架。

电机连接









输出	频率	作动器
MAIN1	400 Hz	右前方，逆时针
MAIN2	400 Hz	左后方，逆时针
MAIN3	400 Hz	左前方，顺时针
MAIN4	400 Hz	右后方，顺时针
AUX1	50 Hz	RC AUX1
AUX2	50 Hz	RC AUX2
AUX3	50 Hz	RC AUX3

参数

- 如果使用默认的四旋翼X型布局的增益，那么在高油门的时候，将会发生姿态震荡。在低油门的时候，较高的增益会获得比较好的响应。这说明，采用基于油门的增益调度可能会提高全油门的响应性能，这可以在mc_att_control中实现。目前，我们仅仅调整增益使得在低油门和高油门的时候不会出现震荡，在低油门的时候达到带宽要求。
 - MC_PITCHRATE_P: 0.05
 - MC_PITCHRATE_D: 0.001
- 电池有6个单元，而不是默认的3个
 - BAT_N_CELLS: 6

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

QAV-R

官网英文原文地址：<http://dev.px4.io/qav-r.html>



Parts List

Frame/ Motors

- QAV-R FPV Racing Quadcopter (5")
- Lumenier 4Power Quick Swap Power Distribution Board
- Lumenier RX2204-14 2300Kv Motor
- Lumenier Mini 20A ESC w/ SimonK AutoShot, 5V/1A BEC (2-4s)
- Gemfan 5x4.5 Nylon Glass Fiber Propeller (Set of 4 - Black)
- Lumenier 1300mAh 3s 60c Lipo Battery (XT60)

FPV Gear

- QAV180/210 Carbon Fiber Vibration Damping Camera Plate (GoPro+Mobius)
- Fat Shark 900TVL CCD FPV Camera (NTSC)
- FatShark 5.8ghz 250mW A/V Transmitter

Accessories

- Professional Travel Case for the QAV-R (nice to have)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

直升机机型

官网英文原文地址：<http://dev.px4.io/airframes-plane.html>

借助于灵活的混控系统，PX4支持包括一般飞机，飞翼，倒V尾飞机以及垂直起降飞机在内的所有能想到的飞机几何构型。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

Wing Wing Z-84

官网英文原文地址：<http://dev.px4.io/airframes-plane-wing-z-84.html>

部件列表

确认订购的是PNF版，该版本包含电机，螺旋桨和电调。Kit版需要单独购买以上部件。

- Zeta Science Wing-Wing Z-84 PNF ([Hobbyking Store](#))
- 1800 mAh 2S LiPo
 - [Team Orion 1800mAh 7.4V 50C 2S1P](#)
- FrSky D4R-II接收机或同类产品（根据手册设置跳帽为PPM输出）
 - [Pixracer kit](#) (包含GPS和电源模块)
 - [Mini telemetry set](#) for HKPilot32
 - [电子空速传感器](#) for HKPilot32 / Pixfalcon
 - 备用零件
 - [O-Rings螺旋桨保护环](#)
 - [备用螺旋桨](#)

舵机连接

输出	频率	作动器
MAIN1	50 Hz	左副翼舵机
MAIN2	50 Hz	右副翼舵机
MAIN3	50 Hz	空
MAIN4	50 Hz	电机控制器

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

垂直起降

官网英文原文地址：<http://dev.px4.io/airframes-vtol.html>

事实上，[PX4飞控系统](#)支持所有的垂直起降机型配置：

- 尾座式 (X/+型布局的双/四旋翼)
- 倾转式 (Firefly Y6)
- 复合式 (飞机+四旋翼)

垂直起降飞行器与其它种类的飞行器共享代码库，所不同的仅仅是增加了额外的控制逻辑，特别是转换阶段。

所有的这些垂直起降飞行器都经过了积极地试飞，并且已经准备好以供使用了。确保为飞行器添加了空速传感器，系统需要根据空速信息决定是否可以安全地进行转换。

关键配置参数

创建新的垂直起降飞行器的配置时，需要正确设置下面这些参数。

- `VT_FW_PERM_STAB` 系统在悬停模式下使用姿态增稳。如果这个参数设置为1，那么在飞机模式下也会使用姿态增稳；如果这个参数设置为0，那么飞机模式下将会使用纯手动飞行。
- `VT_ARSP_TRANS` 参数决定飞行器悬停状态转换到前飞状态的空速阈值，这个值设置过小会导致转换阶段的失速现象。
- `RC_MAP_TRANS_SW` should be assigned to a RC switch before flight. This allows you to check if the multicopter- and fixed wing mode work properly. (Can also be used to switch manually between those two control modes during flight)

尾座式

[构建日志](#)包括更加详细的信息。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

倾转式

[构建日志](#)包括所有的设置以及操作指南，这些会指导整个系统顺利运行。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

复合式

[构建日志](#)包括详细的指南指导如何构建以及复现线面的结果。

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

垂直起降测试

官网英文原文地址：<http://dev.px4.io/airframes-vtol-testing.html>

测试垂直起降飞行器是否正确运行，主要的关注点在转换阶段：

- 工作台
- 飞行中

转换阶段注意事项

目前有3种方式控制垂直起降飞行器进行转换：

- 遥控器切换开关（2挡，辅助1）
- MAVLink指令(MAV_CMD_DO_VTOL_TRANSITION)
- 任务中转换(任务内部执行MAV_CMD_DO_VTOL_TRANSITION)

接收到转换指令后（以上任意方式），垂直起降飞行器进入转换阶段。如果在转换阶段接收到新的转换回原状态的转换指令，系统会立刻切换回原状态。这是在需要的时候放弃转换的安全处理方法。转换完成后，飞行器进入新的状态，此时切换回旧状态的指令将会正常进行。

确保辅助1通道分配到遥控器上的开关，并且空速传感器工作正常。

工作台

移除所有螺旋桨！测试转换是否正常运行，飞行器需要在解锁状态。

默认从多旋翼模式开始：

- 解锁飞行器
- 检查电机是否按照多旋翼配置运行（在滚转/俯仰/偏航输入下方向舵/副翼保持不动）
- 拨动转换开关
- （如果可用）等待转换阶段第一步完成
- 向皮托管吹气模拟空速
- （如果可用）转换阶段第二步将会执行
- 检查电机是否按照固定翼配置运行（滚转/俯仰/偏航输入应该可以控制方向舵/副翼）
- 拨动切换开关
- 观察转换回多旋翼模式
- 检查电机是否按照多旋翼配置运行（在滚转/俯仰/偏航输入下方向舵/副翼保持不动）

飞行中

在测试飞行中转换之前，确保飞行器在多旋翼模式下稳定飞行。总的来说，如果发生异常现象，切换回多旋翼模式使飞行器恢复正常（如果调整的好，飞行器会自动恢复）。

无论如何，飞行中转换需要下面的参数与机型以及操作手技能相匹配：

参数	注释
VT_FW_PERM_STAB	固定翼模式下增稳控制器是否开启
VT_ARSP_BLEND	固定翼控制器激活空速阈值
VT_ARSP_TRANS	完成固定翼模式转换的空速

根据垂直起降飞行器类型的不同，有更多的参数需要调整，查看[参考参数](#)。

手动转换测试

测试手动转换的基本步骤如下所示：

- 在多旋翼模式下解锁并起飞
- 爬升到安全高度（转换后会掉高度）
- 迎风飞行
- 拨动转换开关
- 观察转换过程多旋翼-固定翼
- 在固定翼模式下飞行
- 爬升到安全高度（转换后会掉高度）
- 拨动转换开关
- 观察转换过程固定翼-多旋翼
- 降落并上锁

多旋翼-固定翼

多旋翼到固定翼的转换过程可能会有下面现象发生：

1. 失去控制，并且速度增加（很多因素会造成这种现象）
2. 转换持续时间过长，飞行器在转换完成前飞的太远

对于1)：切换回多旋翼模式（将会立即执行）。尝试识别问题（检查设定点）。对于2)：如果设置了混合空速，并且已经可以作为固定翼控制时飞行器有较高的空速，那么它可能fly around，再给它一些时间完成转换。否则，切换会多旋翼模式并识别问题（检查空速）。

固定翼-多旋翼

从固定翼到多旋翼的转换过程大多数是没问题的。万一失去控制，最好的办法是让它自己恢复。

自动转换测试（任务，指令）

指令转换仅仅可以在自动（任务）模式或者机外飞行模式中执行。确保你有信心在飞行中执行自动/机外模式和转换切换。

切换到手动控制将重新激活转换开关。例如：如果在自动固定翼飞行状态下切换出自动/板外模式，并且转换开关处于多旋翼模式，那么飞行器将立刻转换到多旋翼模式。

步骤

下面的步骤用于测试带有转换过程的任务：

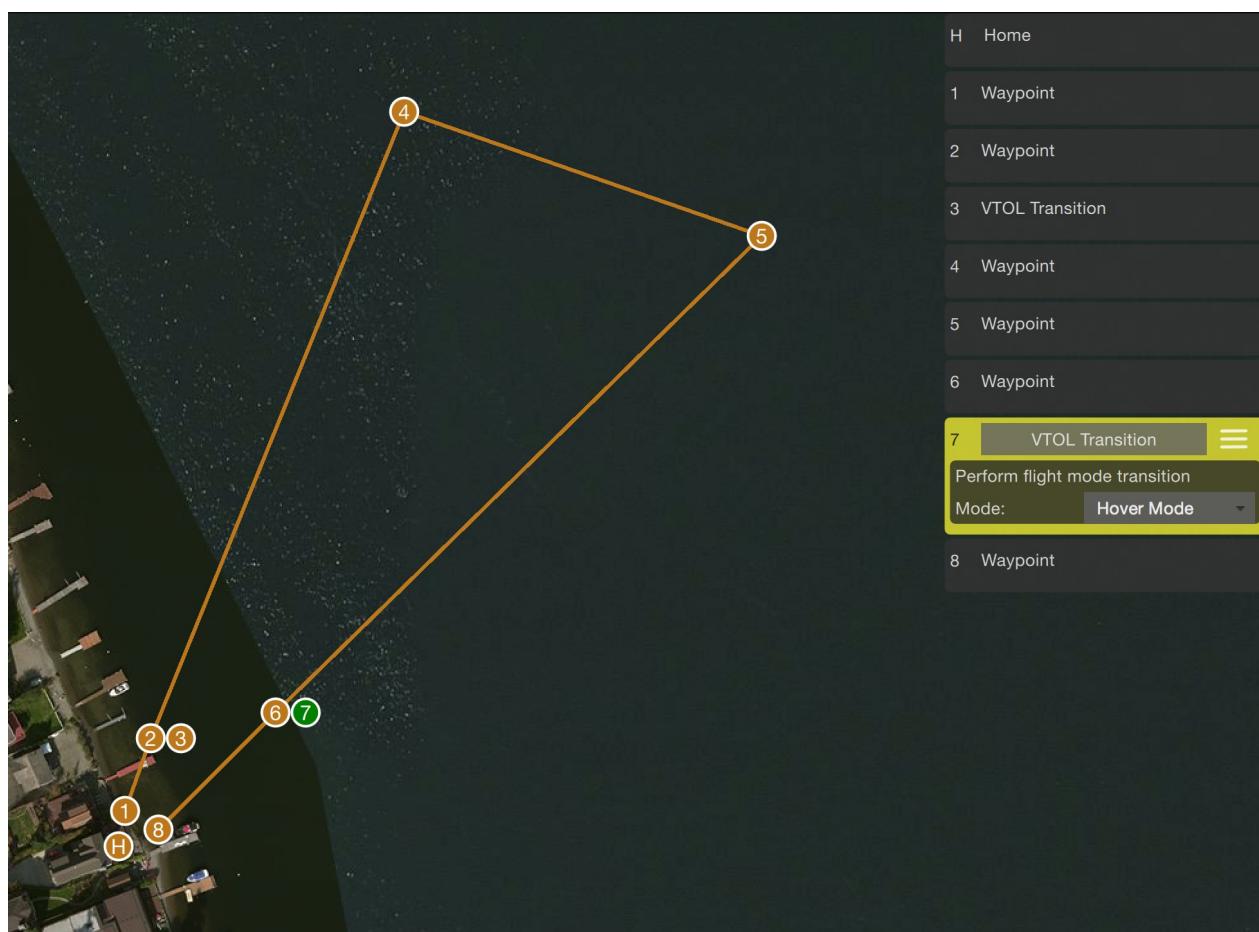
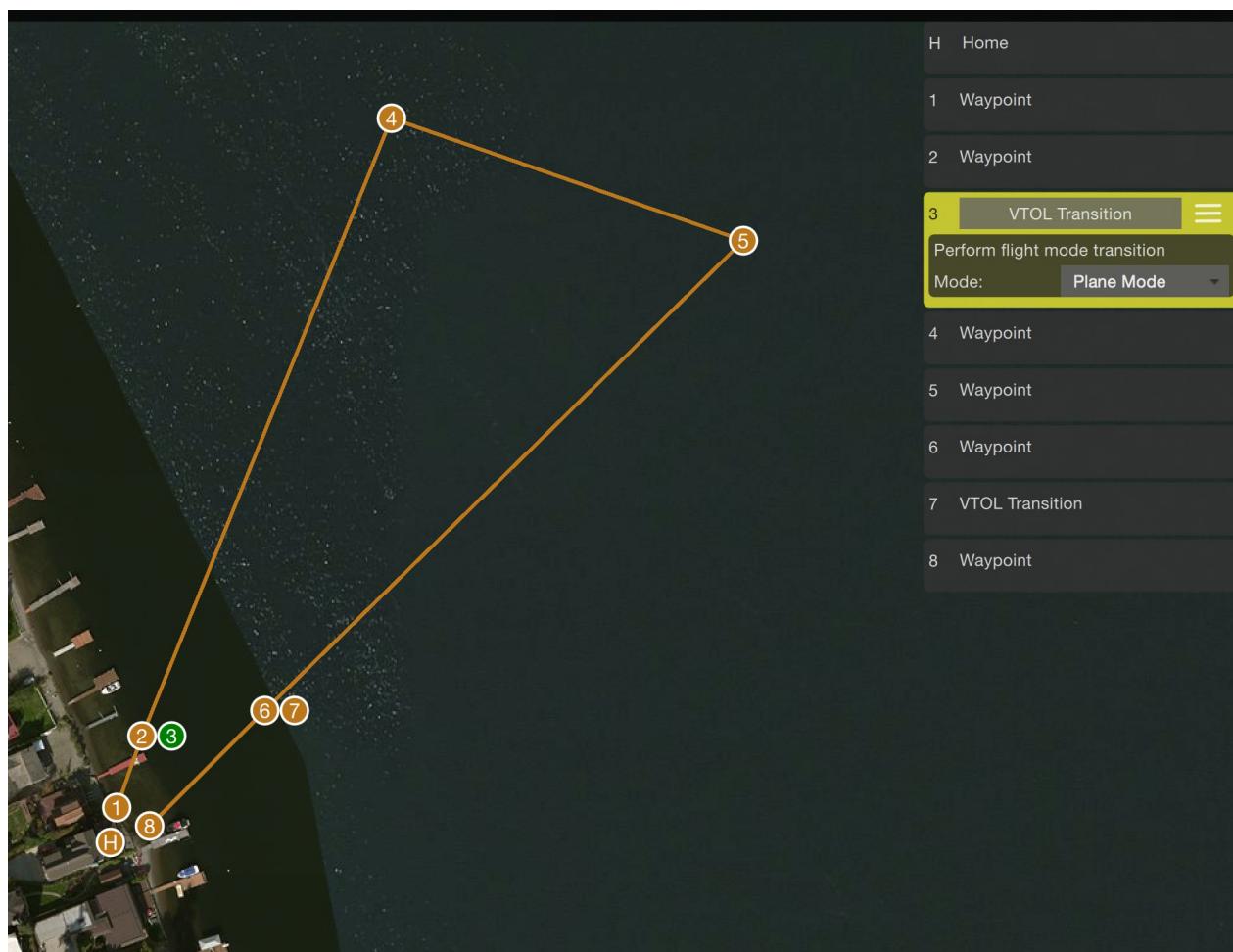
- 上传任务
- 在多旋翼模式下起飞并爬升到任务高度
- 切换到任务模式
- 观察转换到固定翼的过程
- 享受飞行
- 观察转换回多旋翼的过程
- 切出任务模式
- 手动降落

在飞行中，手动切换开关保持在多旋翼位置。如果发生意外，切换回手动模式，飞行器将恢复到多旋翼模式。

任务示例

任务应该至少包含以下内容（也可以看下面的截图）：

- (1) 在起飞位置附近的路径点
- (2) 沿着计划的固定翼飞行路线方向的路径点
- (3) 转换路径点（到固定翼模式）
- (4) 较远的路径点（至少远离转换过程需要的距离）
- (5) 返回的路径点（转换回多旋翼模式需要一段距离，所以设置在距离着陆点一段距离）
- (6) 转换路径点（到多旋翼模式）
- (7) 着陆位置附近的路径点



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

TBS Caipiroshka

官网英文原文地址：<http://dev.px4.io/airframes-vtol-caipiroshka.html>

由TBS Caipirinha稍作修改得到垂直起降型Caipiroshka。

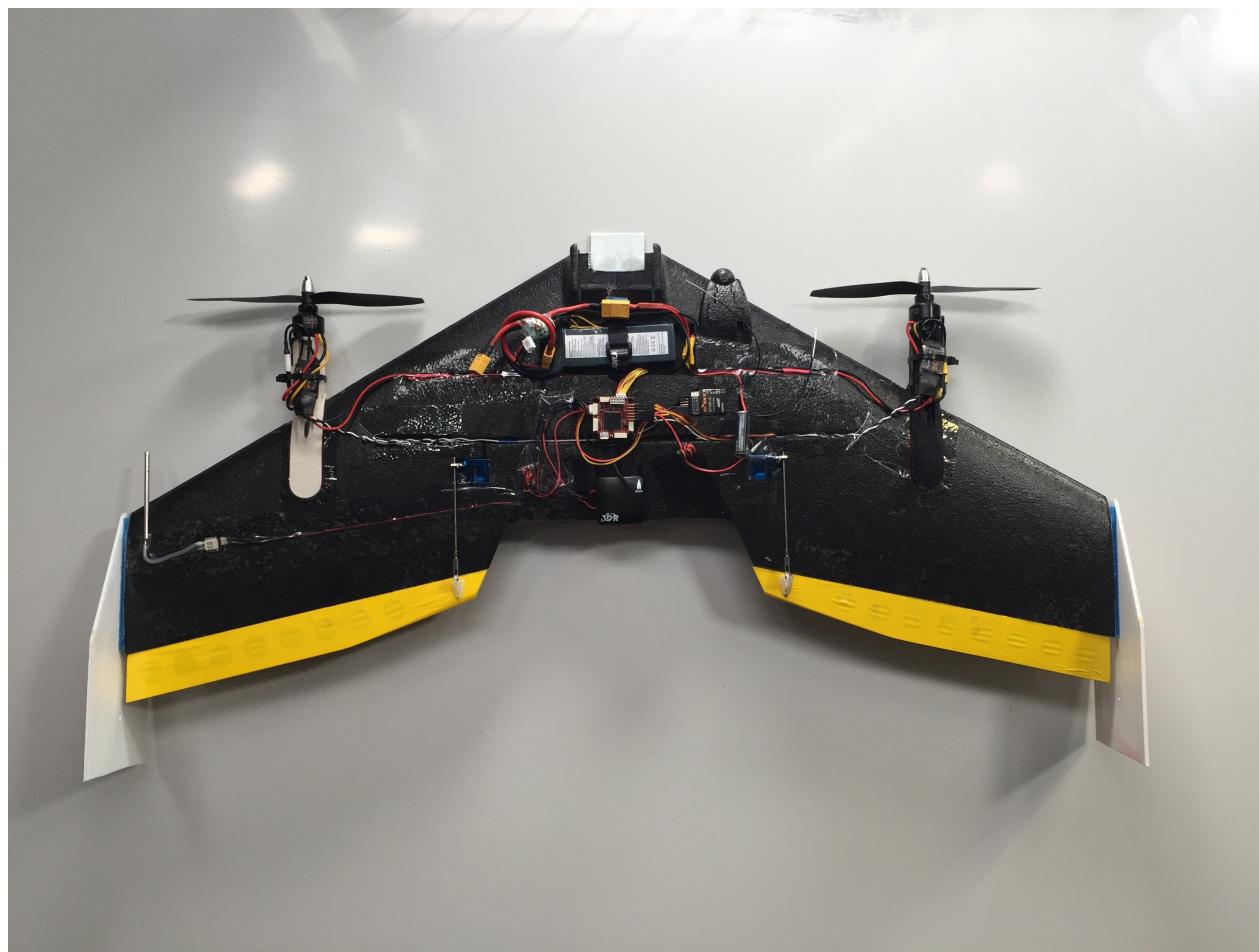
To view this video please enable JavaScript, and consider upgrading to a web browser that supports HTML5 video

部件列表

- TBS Caipirinha飞翼([Eflight商店](#))
- 3D打印的左右电机支架([设计文件](#))
- 顺时针8045螺旋桨([Eflight商店](#))
- 逆时针8045螺旋桨([Eflight商店](#))
- 2个1800 KV 120-180W电机
 - [Quanum MT2208 1800 KV](#)
 - [ePower 2208](#)
- 2个20-30S电调([Eflight商店](#))
- BEC(3A, 5-5.3V) (使用的电调不具备5V电源输出功能时选用)
- 3S 2200 mA锂电池
 - [Team Orion 3S 11.1V 50 C \(Brack商店\)](#)
- [Pixracer飞控板 + 电源模块](#)
- [电子空速传感器](#)

组装

下图展示了组装完成的Caipiroshka的样子



下面是一些关于如何组装该飞行器的提示

飞控板

飞控板应该放置在中间靠近飞行器质心的位置。

电机支架

打印2个电机支架，STL文件在部件列表中已经给出。在机翼的两边各固定1个电机支架，保证电机的轴线大体通过副翼的中心（参照图片）。上图中两个电机支架的水平距离为56cm。在机翼上标记了支架的正确位置后，用透明胶带覆盖支架连接区域的上下表面。然后在该区域涂上一层热熔胶，接着把支架粘到机翼上。在机翼表面与热熔胶之间加入一层透明胶带，可以让你更加容易地拆除电机支架，仅仅撕下胶带即可，同时这样做也不会破坏机翼。当要替换掉损坏的电机支架时，这么做就会非常有用。

电机控制器

电机控制器可以直接用胶水或者扎带固定在电机支架的平坦表面上。可以使用旧烙铁在泡沫表面烧出渠道，以此作为连接到电极的线路。把两个电机的电源线连接到一起，并在末端焊接一个插头，这样做就可以同时连接两个电机和电源模块了。如果电机控制器不能为尾推提供

供5V电源，那么你还需要额外的BEC。

GPS

GPS可以挂载在机身的中后部，这样做有助于机身重心后移，因为2个电机，1个相机，还有可能包括大电池都将使机鼻过重。这样做也可以使GPS远离电源线，有利于减小对外置罗盘的干扰。

空速传感器

皮托管需要固定在机翼外侧以保证不受螺旋桨气流的干扰。皮托管到电机轴线的水平距离应该大于螺旋桨的半径。同样的，用旧烙铁为皮托管以及空速传感器划出凹槽（参照图片），并划出到其它部件的连接渠道。

I2C设备

空速传感器和外置罗盘（在GPS外罩里）都需要连接到飞控板的I2C总线，因此，还需要一个I2C分线板。把分线板连接到飞控板的I2C总线上，然后用标准I2C线把外置罗盘和空速传感器连接到分线板上。在上图中，分线板位于GPS模块的左侧。

副翼

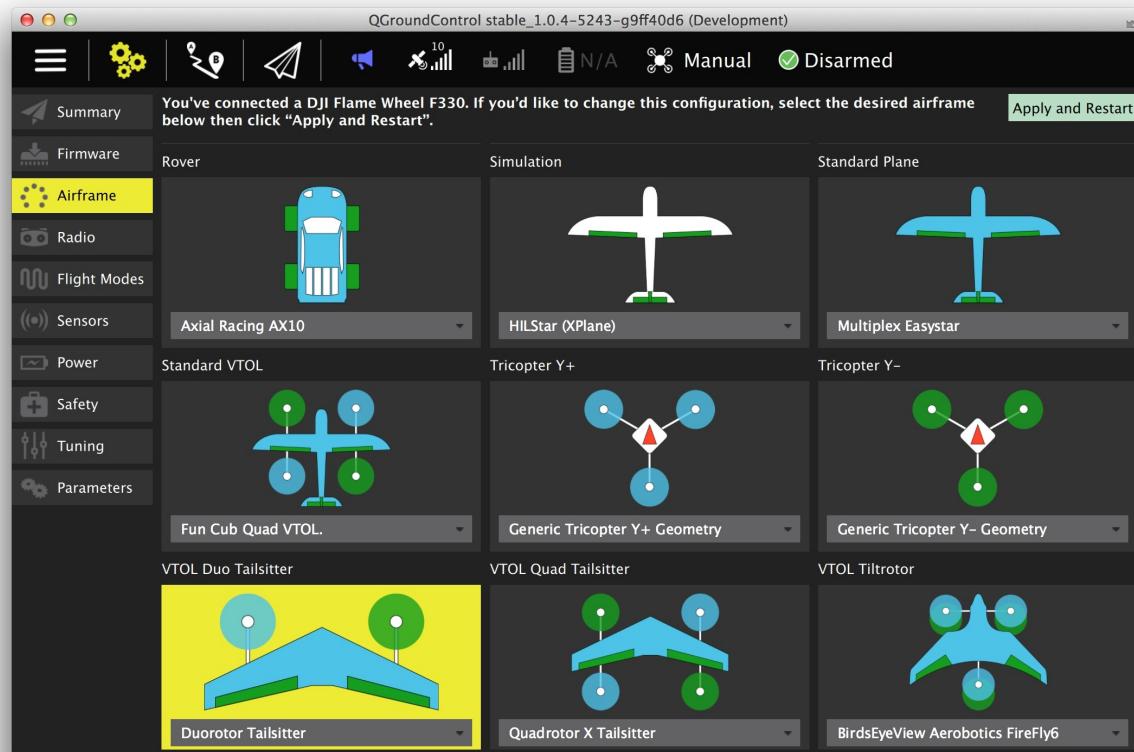
使用透明胶带把副翼连接到机翼的后部，可以参照Blacksheep团队在TBS Caiprinha组装手册中给出的操作指南。

通用组装原则

在把所有部件固定到机身之前，先用透明胶带把它们暂时固定在合适的位置并检查重心是否在TBS Caiprinha组装手册中推荐的范围内。根据你想要添加的额外部件（GoPro或者大电池），调整这些部件的位置。

机型配置

在[QGroundControl](#)中切换到配置页面并打开机型选项卡，滚动列表寻找垂直起降双旋翼尾座型图标，在下拉菜单中选择 Duorotor Tailsitter



舵机连接

下表中的描述假设飞行器平放在桌子上，机腹向下，用户位于飞行器前方，面对飞行器。

输出	频率	作动器
MAIN1	400 Hz	左电机
MAIN2	400 Hz	右电机
MAIN3	400 Hz	空
MAIN4	400 Hz	空
MAIN5	50 Hz	左副翼
MAIN6	50 Hz	右副翼

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

船舶，潜水艇，飞艇，车辆

官网英文原文地址：<http://dev.px4.io/airframes-experimental.html>

实验飞行器和通用机器人

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

协同计算机

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

Pixhawk系列飞控板的协同计算机

官网英文原文地址：<http://dev.px4.io/pixhawk-companion-computer.html>

无论何种协同计算机（Raspberry Pi, Odroid, Tegra K1），与Pixhawk系列飞控板之间的接口是相同的：它们通过串口连接到Pixhawk上的TELEM2，这个端口专用于与协同计算机相连。连接的消息格式是MAVLink。

Pixhawk设置

参考下表，设置SYS_COMPANION参数（System参数组）

变更参数后需要重启飞控使其生效。

- 0：禁用TELEM2上的MAVLink输出（默认）
- 921600：使能MAVLink输出，波特率：921600, 8N1（推荐）
- 157600：使能MAVLink输出，OSD模式，波特率：57600
- 257600：使能MAVLink输出，监听模式，波特率：57600

协同计算机设置

为了能够接收MAVLink消息，协同计算机需要运行一些和串口通讯的软件，最常用的是：

- MAVROS：ROS
- C/C++ example code：自定义的代码
- MAVProxy：在串口和UDP之间传输MAVLink

硬件设置

根据下面的说明连接串口。所有Pixhawk串口工作在3.3V，兼容5V。

许多现代协同计算机在UART端口仅支持1.8V的电压，并且可能在3.3V下损坏。使用电压转换器。大多数时候，可以使用的硬件串口有特定的功能（modem or console），在使用之前，需要在Linux下重新配置它们。

安全的做法是使用FTDI（USB转串口适配器），并按照下面说明连接它。这大多数时候都管用并且很容易设置。

	TELEM2		FTDI	
1	+5V (red)			DO NOT CONNECT!
2	Tx (out)	5		FTDI RX (yellow) (in)
3	Rx (in)	4		FTDI TX (orange) (out)
4	CTS (in)	6		FTDI RTS (green) (out)
5	RTS (out)	2		FTDI CTS (brown) (in)
6	GND	1		FTDI GND (black)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

使用**DroneKit**的机器人

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

使用DroneKit与PX4通讯

官网英文原文地址：<http://dev.px4.io/dronekit-example.html>

DroneKit可以帮助创建强大的无人机应用。这些应用运行在无人机的协同计算机上，通过执行计算密集但又需要低延迟的任务（计算机视觉）来增强飞控计算机。

DroneKit和PX4目前致力于获得完全兼容。截止DroneKit-python 2.2.0，仅提供任务处理和状态监控这样的基本支持。

配置DroneKit

首先，从当前主分支安装DroneKit-python

```
git clone https://github.com/dronekit/dronekit-python.git
cd ./dronekit-python
sudo python setup.py build
sudo python setup.py install
```

创建一个新的python文件并导入DroneKit, pymavlink和基本模块

```
# Import DroneKit-Python
from dronekit import connect, Command, LocationGlobal
from pymavlink import mavutil
import time, sys, argparse, math
```

连接到无人机或模拟器的MAVLink端口

```
# Connect to the Vehicle
print "Connecting"
connection_string = '127.0.0.1:14540'
vehicle = connect(connection_string, wait_ready=True)
```

显示一些基本的状态信息

```
# Display basic vehicle state
print " Type: %s" % vehicle._vehicle_type
print " Armed: %s" % vehicle.armed
print " System status: %s" % vehicle.system_status.state
print " GPS: %s" % vehicle.gps_0
print " Alt: %s" % vehicle.location.global_relative_frame.alt
```

完整的任务范例

下面的python脚本文件给出了使用DroneKit和PX4的完整任务范例。目前还不完全支持模式切换，因此我们发送自定义的模式切换指令。

```
#####
# @File DroneKitPX4.py
# Example usage of DroneKit with PX4
#
# @author Sander Smeets <sander@droneslab.com>
#
# Code partly based on DroneKit (c) Copyright 2015-2016, 3D Robotics.
#####

# Import DroneKit-Python
from dronekit import connect, Command, LocationGlobal
from pymavlink import mavutil
import time, sys, argparse, math

#####
# Settings
#####

connection_string      = '127.0.0.1:14540'
MAV_MODE_AUTO    = 4
# https://github.com/PX4/Firmware/blob/master/Tools/mavlink_px4.py

#
# Parse connection argument
parser = argparse.ArgumentParser()
parser.add_argument("-c", "--connect", help="connection string")
args = parser.parse_args()

if args.connect:
    connection_string = args.connect

#####
# Init
#####

# Connect to the Vehicle
print "Connecting"
vehicle = connect(connection_string, wait_ready=True)

def PX4setMode(mavMode):
    vehicle._master.mav.command_long_send(vehicle._master.target_system, vehicle._master.target_component,
                                           mavutil.mavlink.MAV_CMD_DO_SET_MODE, 0,
                                           mavMode,
                                           0, 0, 0, 0, 0, 0)
```

```

def get_location_offset_meters(original_location, dNorth, dEast, alt):
    """
    Returns a LocationGlobal object containing the latitude/longitude `dNorth` and `dEast`
    specified `original_location`. The returned Location has the same `alt` value
    as `original_location`.

    The function is useful when you want to move the vehicle around specifying locations
    the current vehicle position.

    The algorithm is relatively accurate over small distances (10m within 1km) except clo
    For more information see:
    http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longi
    """
    earth_radius=6378137.0 #Radius of "spherical" earth
    #Coordinate offsets in radians
    dLat = dNorth/earth_radius
    dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position in decimal degrees
    newlat = original_location.lat + (dLat * 180/math.pi)
    newlon = original_location.lon + (dLon * 180/math.pi)
    return LocationGlobal(newlat, newlon,original_location.alt+alt)

#####
# Listeners
#####

home_position_set = False

#Create a message listener for home position fix
@vehicle.on_message('HOME_POSITION')
def listener(self, name, home_position):
    global home_position_set
    home_position_set = True

#####
# Start mission example
#####

# wait for a home position lock
while not home_position_set:
    print "Waiting for home position..."
    time.sleep(1)

# Display basic vehicle state
print " Type: %s" % vehicle._vehicle_type

```

```

print " Armed: %s" % vehicle.armed
print " System status: %s" % vehicle.system_status.state
print " GPS: %s" % vehicle.gps_0
print " Alt: %s" % vehicle.location.global_relative_frame.alt

# Change to AUTO mode
PX4setMode(MAV_MODE_AUTO)
time.sleep(1)

# Load commands
cmds = vehicle.commands
cmds.clear()

home = vehicle.location.global_frame

# takeoff to 10 meters
wp = get_location_offset_meters(home, 0, 0, 10);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# move 10 meters north
wp = get_location_offset_meters(wp, 10, 0, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# move 10 meters east
wp = get_location_offset_meters(wp, 0, 10, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# move 10 meters south
wp = get_location_offset_meters(wp, -10, 0, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# move 10 meters west
wp = get_location_offset_meters(wp, 0, -10, 0);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# land
wp = get_location_offset_meters(home, 0, 0, 10);
cmd = Command(0,0,0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_C
cmds.add(cmd)

# Upload mission
cmds.upload()
time.sleep(2)

# Arm vehicle
vehicle.armed = True

# monitor mission execution

```

```
nextwaypoint = vehicle.commands.next
while nextwaypoint < len(vehicle.commands):
    if vehicle.commands.next > nextwaypoint:
        display_seq = vehicle.commands.next+1
        print "Moving to waypoint %s" % display_seq
        nextwaypoint = vehicle.commands.next
    time.sleep(1)

# wait for the vehicle to land
while vehicle.commands.next > 0:
    time.sleep(1)

# Disarm vehicle
vehicle.armed = False
time.sleep(1)

# Close vehicle object before exiting script
vehicle.close()
time.sleep(1)
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

使用**ROS**的机器人

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

外部控制

官网英文原文地址：<http://dev.px4.io/offboard-control.html>

警告：外部控制是很危险的。在进行外部控制飞行之前，开发者需要保证有充分的准备、测试以及安全预防措施。

外部控制允许使用运行在飞控板外部的软件去控制px4飞行控制栈。通过MAVLink协议完成这些操作，特别是`SET_POSITION_TARGET_LOCAL_NED`和`SET_ATTITUDE_TARGET`消息。

外部控制固件设置

在开始外部控制开发之前，固件方面需要做两项设置。

1. 映射一个**RC**切换开关为外部模式激活开关

在QGroundcontrol中载入参数，并设置`RC_MAP_OFFB_SW`参数为想要控制外部模式激活的RC通道。这样做是非常有用的，当在外部模式出现问题时可以切换到位置控制模式。

尽管这一步并不是强制的，因为通过MAVLink消息同样可以激活外部模式。但是我们认为这种方式更加安全。

2. 使能协同计算机接口

将参数`SYS_COMPANION`设置为921600（推荐）或者57600。这个参数将会以合适的波特率(921600 8N1或者57600 8N1)激活TELEM2端口上的MAVLink消息流，这与内部模式的数据流是相同的。

有关这些数据流的更多信息，参考[source code](#)中的"MAVLINK_MODE_ONBOARD"。

硬件设置

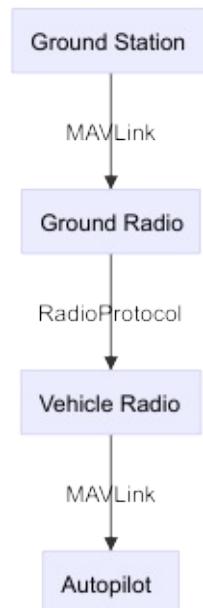
通常，有3种方式配置板外通讯

1. 数传

1. 一个连接到飞控板的UART端口
2. 一个连接到地面站计算机

参考数传包括：

- Lairdtech RM024
- Digi International XBee Pro



2. 机载协同计算机

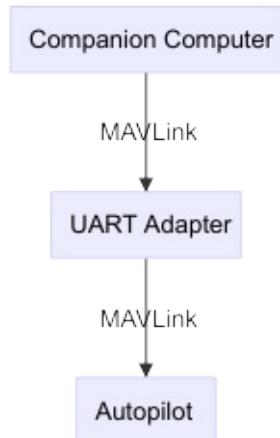
一个挂载在飞行器上的小型计算机，通过串口转USB适配器连接到飞控板。有许多可用的选择，主要取决于除了向飞控板发送指令外还想要进行的额外操作。

低性能机载计算机：

- Odroid C1+ or Odroid XU4
- Raspberry Pi
- Intel Edison

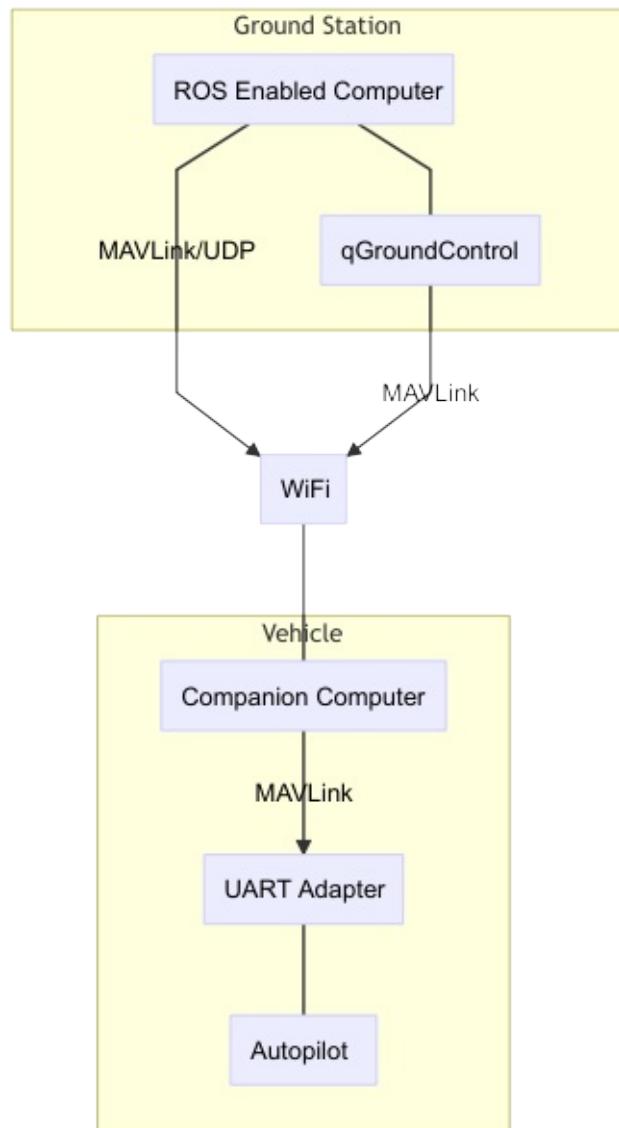
高性能机载计算机：

- Intel NUC
- Gigabyte Brix
- Nvidia Jetson TK1



3. 机载计算机和到ROS的WIFI连接（推荐）

一个挂载在飞行器上的小型计算机，通过串口转USB适配器连接到飞控板，同时提供到运行ROS的地面站的WIFI连接。可以是上一部分的任意一个机载计算机，同时再加一个WIFI适配器。例如：Intel NUC D34010WYB有一个PCI Express Half-Mini接口，可以连接一个[Intel Wifi Link 5000](#)适配器。



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

在 Raspberry Pi (树莓派) 上安装ROS

官网英文原文地址：<http://dev.px4.io/ros-raspberrypi-installation.html>

本文介绍如何在一个作为 Pixhawk 协同计算机的树莓派 2 上安装 ROS-indigo。

准备

- 一个可以工作的树莓派，配有监视器、键盘或者配置好的SSH连接。
- 这份指南假定你已经在树莓派上安装好了 Raspbian "JESSIE"，如果没有，[安装它](#) 或者 [升级 Raspbian Wheezy 到 Jessie](#)。

安装

参照[指南](#)安装ROS Indigo。注意：安装"ROS-Comm"版本，"Desktop"版本太过庞大。

安装可能遇到的错误

如果下载包（例如 `sudo apt-get install ros-indigo-ros-tutorials`）时遇到错误"unable to locate package ros-indigo-ros-tutorials"，那么按照下面方法操作：

进入你的catkin工作空间（例如`~/ros_catkin_ws`），并修改包的名字

```
$ cd ~/ros_catkin_ws
$ rosinstall_generator ros_tutorials --rosdistro indigo --deps --wet-only --exclude roslib
[1] [2]
```

接着，用wstool升级你的工作空间

```
$ wstool merge -t src indigo-custom_ros.rosinstall
$ wstool update -t src
```

最后(仍然在工作空间文件夹), source 并构建你的文件。

```
$ source /opt/ros/indigo/setup.bash  
$ source devel/setup.bash  
$ catkin_make
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

MAVROS

官网英文原文地址：<http://dev.px4.io/ros-mavros-installation.html>

MAVROSROS包允许在运行ROS的计算机、支持MAVLink的飞控板以及支持MAVLink的地面上站之间通讯。虽然**MAVROS**可以用来与任何支持MAVLink的飞控板通讯，但是本文仅就PX4飞行栈与运行ROS的协同计算机之间的通讯予以说明。

安装

MAVROS可以通过源文件或者二进制文件安装。推荐使用源文件安装。

二进制文件安装（Debian / Ubuntu）

从v0.5开始有x86和amd64平台的预编译的Debian安装包，从v0.9版本开始有Ubuntu armhf平台上的ARMv7安装包。

使用 `apt-get` 安装即可：

```
$ sudo apt-get install ros-indigo-mavros ros-indigo-mavros-extras
```

源文件安装

依赖

假定你有一个catkin工作空间位于 `~/catkin_ws`，如果没有，则创建一个：

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin init
```

安装MAVROS需要用到ROS python工具 `wstool`，`rosinstall` 和 `catkin_tools`，尽管在安装ROS时可能已经安装过这些工具，但是仍然可以重新安装一遍：

```
$ sudo apt-get install python-wstool python-rosinstall-generator python-catkin-tools
```

注意，尽管可以使用`catkin_make`构建这些包，但是推荐使用`catkin_tools`，因为它更加友好，功能更加全面。

如果这是第一次使用wstool，那么需要初始化你的源空间：

```
$ wstool init ~/catkin_ws/src
```

现在准备构建

```
# 1. get source (upstream - released)
$ rosinstall_generator --upstream mavros | tee /tmp/mavros.rosinstall
    # alternative: latest source
$ rosinstall_generator --upstream-development mavros | tee /tmp/mavros.rosinstall

# 2. get latest released mavlink package
# you may run from this line to update ros-*-*mavlink package
$ rosinstall_generator mavlink | tee -a /tmp/mavros.rosinstall

# 3. Setup workspace & install deps
$ wstool merge -t src /tmp/mavros.rosinstall
$ wstool update -t src
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y

# finally - build
$ catkin build
```

提示:如果在树莓派上安装MAVROS，当运行 `rosdep install ...` 时可能会遇到和操作系统有关的错误。在rosdep命令中加上 `--os=OS_NAME:OS_VERSION`，其中OS_NAME是你的操作系统名称，OS_VERSION是你的操作系统版本（例如：`--os=debian:jessie`）

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22

12:56:03

MAVROS外部控制例程

官网英文原文地址：<http://dev.px4.io/ros-mavros-offboard.html>

外部控制是危险的。如果在真机上操作，确保可以在出错的时候切换回手动控制。

下面的教程是基本的外部控制，通过MAVROS应用在Gazebo模拟的Iris四旋翼上。在教程最后，应该会得到与下面视频相同的结果，即缓慢起飞到高度2米。

....

代码

在ROS包中创建offb_node.cpp文件，并粘贴下面内容：

```
/*
 * @file offb_node.cpp
 * @brief offboard example node, written with mavros version 0.14.2, px4 flight
 * stack and tested in Gazebo SITL
 */

#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>

mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "offb_node");
    ros::NodeHandle nh;

    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
        ("mavros/state", 10, state_cb);
    ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
        ("mavros/setpoint_position/local", 10);
    ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>
        ("mavros/cmd/arming");
    ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
        ("mavros/set_mode");

    //the setpoint publishing rate MUST be faster than 2Hz
```

```

ros::Rate rate(20.0);

// wait for FCU connection
while(ros::ok() && current_state.connected){
    ros::spinOnce();
    rate.sleep();
}

geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;

//send a few setpoints before starting
for(int i = 100; ros::ok() && i > 0; --i){
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}

mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";

mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){
    if( current_state.mode != "OFFBOARD" &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        if( set_mode_client.call(offb_set_mode) &&
            offb_set_mode.response.success){
            ROS_INFO("Offboard enabled");
        }
        last_request = ros::Time::now();
    } else {
        if( !current_state.armed &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( arming_client.call(arm_cmd) &&
                arm_cmd.response.success){
                ROS_INFO("Vehicle armed");
            }
            last_request = ros::Time::now();
        }
    }
}

local_pos_pub.publish(pose);

ros::spinOnce();
rate.sleep();
}

```

```
    return 0;
}
```

提示：本过程需要对ROS有一定的了解。创建工作空间后需要 `source devel/setup.bash`，否则会出现找不到package的情况，要想保证工作空间已配置正确需确保`ROS_PACKAGE_PATH`环境变量包含你的工作空间目录，采用 `echo $ROS_PACKAGE_PATH` 命令查看是否包含了你创建的package的路径，此操作也可以通过直接在`.bashrc`文件最后添加路径的方式解决。

代码解释

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
```

`mavros_msgs` 包含所有用于MAVROS服务和主题的自定义消息。所有服务和主题以及它们所对应的消息类型参照文档[mavros wiki](#)。

```
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}
```

创建一个简单的回调函数，它可以保存飞控的当前状态。我们可以用它检查连接状态，解锁状态以及外部控制标志。

```
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/state", 10, state_cb)
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("mavros/setpoint_
ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd
ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>("mavros/set_m
```

我们实例化一个用来发布指令位置的发布器，一个请求解锁的客户端和一个请求改变模式的客户端。注意，对你自己的系统，根据启动文件中节点名字的不同，“mavros”前面的部分会有所不同。

```
//the setpoint publishing rate MUST be faster than 2Hz
ros::Rate rate(20.0);
```

px4飞行栈在外部控制指令之间有500ms的时限，如果超过了时限，那么飞控将会切换回进入外部控制模式之前的模式。这正是考虑可能的延迟，发布频率必须高于2Hz的原因。这同样也是推荐从位置控制模式进入外部控制模式的原因，如果外部控制模式发生故障，飞行器将会停止动作并处于盘旋状态。

```
// wait for FCU connection
while(ros::ok() && current_state.connected){
    ros::spinOnce();
    rate.sleep();
}
```

在发布之前，需要等待MAVROS和飞控建立连接。一旦接收到心跳包，该循环就会立即退出。

```
geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;
```

即使px4飞行栈工作在航空常用的NED坐标系，MAVROS仍然会将这些坐标转换到标准的ENU坐标系，反之亦然。这是我们将Z设置为+2的原因。

```
//send a few setpoints before starting
for(int i = 100; ros::ok() && i > 0; --i){
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}
```

在进入外部控制模式之前，就必须开始发布指令，否则模式切换会被拒绝。这里，100是个随意选取的值。

```
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";
```

设置自定义模式为 OFFBOARD，参考支持的[模式列表](#)

```

mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){
    if( current_state.mode != "OFFBOARD" &&
        (ros::Time::now() - last_request > ros::Duration(5.0))){
        if( set_mode_client.call(offb_set_mode) &&
            offb_set_mode.response.success){
            ROS_INFO("Offboard enabled");
        }
        last_request = ros::Time::now();
    } else {
        if( !current_state.armed &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( arming_client.call(arm_cmd) &&
                arm_cmd.response.success){
                ROS_INFO("Vehicle armed");
            }
            last_request = ros::Time::now();
        }
    }

    local_pos_pub.publish(pose);

    ros::spinOnce();
    rate.sleep();
}

```

剩下的代码比较好理解。在解锁并起飞后，不断地请求切换至外部控制模式。在请求之间间隔5秒，不至于让飞控响应不过来。在同样的循环里，以合适的频率持续发送位姿指令。

提示:出于解释的目的，这份代码经过了简化。在更大的系统中，创建一个新的负责周期性发送目标指令的线程往往更加有用。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

使用视觉或运动捕捉系统

官网英文原文地址：<http://dev.px4.io/external-position.html>

本文旨在使用除GPS之外的位置数据源构建一个基于PX4的系统，例如像VICON、Optitrack之类的运动捕捉系统和像ROVIO、SVO或者PTAM之类的基于视觉的估计系统。

位置估计既可以来源于板载计算机，也可以来源于外部系统（例如：VICON）。这些数据用于更新机体相对于本地坐标系的位置估计。来自于视觉或者运动捕捉系统的朝向信息也可以被适当整合进姿态估计器中。

现在，这个系统被用来进行室内位置控制或者基于视觉的路径点导航。

对于视觉，用来发送位姿数据的MAVLink消息是VISION_POSITION_ESTIMATE。对于运动捕捉系统，相应的则为ATT_POS_MOCAP。

默认发送这些消息的应用是ROS-Mavlink接口MAVROS，当然，也可以直接使用纯C/C++代码或者MAVLink()库来发送它们。

使能外部位姿输入

需要设置两个参数（从QGroundControl或者NSH shell）来使能或者禁用视觉/运动捕捉。

设置系统参数 CBRK_NO_VISION 为0来使能视觉位置估计。

设置系统参数 ATT_EXT_HDG_M 为1或者2来使能外部朝向估计。设置为1使用视觉，设置为2使用运动捕捉。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

OctoMap

官网英文原文地址：<http://dev.px4.io/simulation-gazebo-octomap.html>

[OctoMap库](#)实现了一个三维占据栅格地图的方法。本文介绍如何在[RotorS仿真](#)中使用它。

安装

需要预先安装ROS，Gazebo和Rotors Simulator插件，按照Rotors Simulator中的[指南](#)安装这些。

接着，安装OctoMap库

```
sudo apt-get install ros-indigo-octomap ros-indigo-octomap-mapping
rosdep install octomap_mapping
rosmake octomap_mapping
```

现在，打开`~/catkin_ws/src/rotors_simulator/rotors_gazebo/CMakeLists.txt`并在文件底部添加下面内容：

```
find_package(octomap REQUIRED)
include_directories(${OCTOMAP_INCLUDE_DIRS})
link_libraries(${OCTOMAP_LIBRARIES})
```

打开`~/catkin_ws/src/rotors_simulator/rotors_gazebo/package.xml`添加下面内容：

```
<build_depend>octomap</build_depend>
<run_depend>octomap</run_depend>
```

执行下面两行

提示：第一行是将默认的shell编辑器（vim）修改为gedit。推荐不熟悉vim的用户使用，如果熟悉的话，可以忽略。

```
export EDITOR='gedit'
rosed octomap_server octomap_tracking_server.launch
```

将下面两行

```
<param name="frame_id" type="string" value="map" />
...
<!--remap from="cloud_in" to="/rgbdslam/batch_clouds" /-->
```

修改为

```
<param name="frame_id" type="string" value="world" />
...
<remap from="cloud_in" to="/firefly/vi_sensor/camera_depth/depth/points" />
```

运行仿真

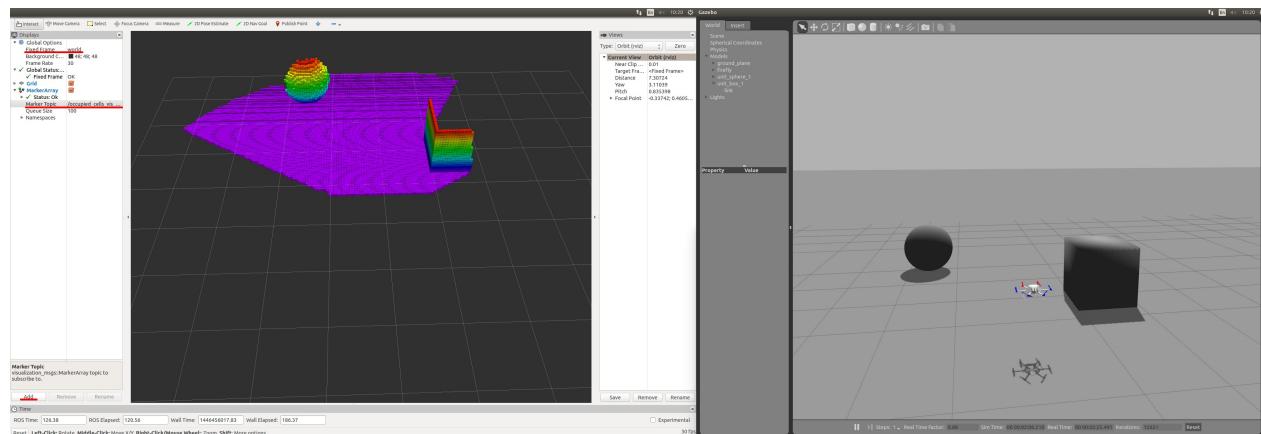
现在，在三个不同的终端窗口中执行下面三行。这将打开Gazebo，Rviz和octomap服务器。

```
roslaunch rotors_gazebo mav_hovering_example_with_vi_sensor.launch mav_name:=firefly
rviz
roslaunch octomap_server octomap_tracking_server.launch
```

在Rviz窗口的左上方，修改域 Fixed Frame ，将 map 改为 world ，然后在窗口左下方单击add按钮并选择MarkerArray，最后双击MarkerArray，并将 Marker Topic 从 /free_cells_vis_array 修改为 /occupied_cells_vis_array 。

现在，你应该看到地面的一部分。

在Gazebo窗口的红色旋翼飞行器前方插入一个立方体，此时你应该可以在Rviz中看到它。



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22
12:56:03

传感器和执行机构总线

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

I2C

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

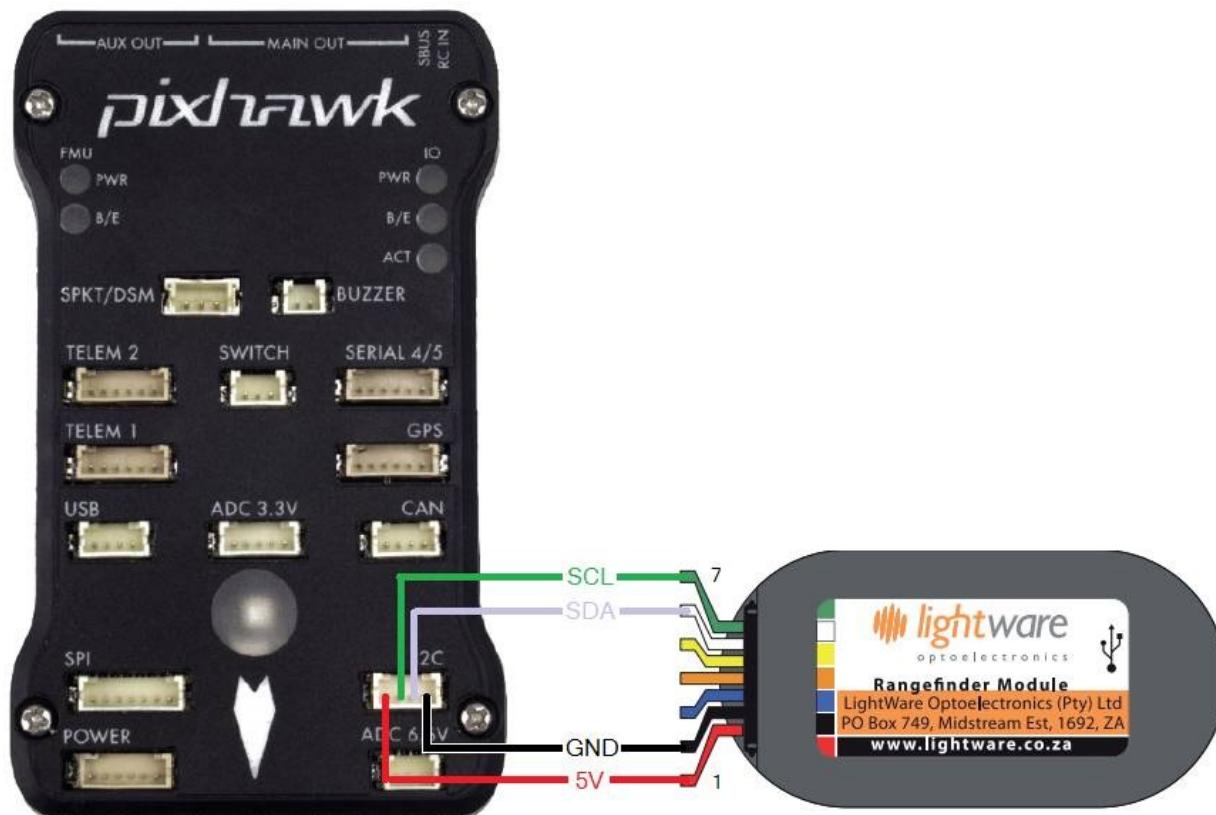
Lightware SF1XX lidar setup

官网英文原文地址：http://dev.px4.io/sf1xx_lidar_setup.html

This page shows you how to set up one of following lidars:

1. SF10/a
2. SF10/b
3. SF10/c
4. SF11/c

Driver supports only i2c connection.



Configuring lidar

You should connect to sensor via usb (it has internal usb to serial converter), run terminal, press `space` and check that i2c address equal to `0x66`.

Newer sensor versions already have `0x66` preconfigured. Older have `0x55` which conflicts with `rgbled` module.

Configuring PX4

Use the `SENS_EN_SF1XX` parameter to select the lidar model and then reboot.

- `0` lidar disabled
- `1` SF10/a
- `2` SF10/b
- `3` SF10/c
- `4` SF11/c

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

UAVCAN Introduction

官网英文原文地址：<http://dev.px4.io/uavcan-intro.html>



[UAVCAN](#) is an onboard network which allows the autopilot to connect to avionics. It supports hardware like:

- Motor controllers
 - [Pixhawk ESC](#)
 - [SV2740 ESC](#)
- Airspeed sensors
 - [Thiemar airspeed sensor](#)
- GNSS receivers for GPS and GLONASS
 - [Zubax GNSS](#)

In contrast to hobby-grade devices it uses rugged, differential signalling and supports firmware upgrades over the bus. All motor controllers provide status feedback and implement field-oriented-control (FOC).

Upgrading Node Firmware

The PX4 middleware will automatically upgrade firmware on UAVCAN nodes if the matching firmware is supplied. The process and requirements are described on the [UAVCAN Firmware](#) page.

Enumerating and Configuring Motor Controllers

The ID and rotational direction of each motor controller can be assigned after installation in a simple setup routine: [UAVCAN Node Enumeration](#). The routine can be started by the user through QGroundControl.

Useful links

- [Homepage](#)
- [Specification](#)
- [Implementations and tutorials](#)

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22
12:56:03

UAVCAN Bootloader

官网英文原文地址：<http://dev.px4.io/uavcan-bootloader-installation.html>

UAVCAN Bootloader Installation

UAVCAN devices typically ship with a bootloader pre-installed. Do not follow the instructions in this section unless you are developing UAVCAN devices.

Overview

The PX4 project includes a standard UAVCAN bootloader for STM32 devices.

The bootloader occupies the first 8–16 KB of flash, and is the first code executed on power-up. Typically, the bootloader performs low-level device initialization, automatically determines the CAN bus baud rate, acts as a UAVCAN dynamic node ID client to obtain a unique node ID, and waits for confirmation from the flight controller before proceeding with application boot.

This process ensures that a UAVCAN device can recover from invalid or corrupted application firmware without user intervention, and also permits automatic firmware updates.

Prerequisites

Installing or updating the UAVCAN bootloader requires:

- An SWD or JTAG interface (depending on device), for example the [BlackMagic Probe](#) or the [ST-Link v2](#);
- An adapter cable to connect your SWD or JTAG interface to the UAVCAN device's debugging port;
- A [supported ARM toolchain](#).

Device Preparation

If you are unable to connect to your device using the instructions below, it's possible that firmware already on the device has disabled the MCU's debug pins. To recover from this, you will need to connect your interface's NRST or nSRST pin (pin 15 on the standard ARM 20-pin connector) to your MCU's NRST pin. Obtain your device schematics and PCB layout or contact the manufacturer for details.

Installation

After compiling or obtaining a bootloader image for your device (refer to device documentation for details), the bootloader must be copied to the beginning of the device's flash memory.

The process for doing this depends on the SWD or JTAG interface used.

BlackMagic Probe

Ensure your BlackMagic Probe [firmware is up to date](#).

Connect the probe to your UAVCAN device, and connect the probe to your computer.

Identify the probe's device name. This will typically be `/dev/ttyACM<x>` or `/dev/ttyUSB<x>`.

Power up your UAVCAN device, and run:

```
arm-none-eabi-gdb /path/to/your/bootloader/image.elf
```

At the `gdb` prompt, run:

```
target extended /dev/ttyACM0
monitor connect_srst enable
monitor swdp_scan
attach 1
set mem inaccessible-by-default off
load
run
```

If `monitor swdp_scan` returns an error, ensure your wiring is correct, and that you have an up-to-date version of the BlackMagic firmware.

ST-Link v2

Ensure you have a recent version—at least 0.9.0—of [OpenOCD](#).

Connect the ST-Link to your UAVCAN device, and connect the ST-Link to your computer.

Power up your UAVCAN device, and run:

```
openocd -f /path/to/your/openocd.cfg &
arm-none-eabi-gdb /path/to/your/bootloader/image.elf
```

At the `gdb` prompt, run:

```
target extended-remote localhost:3333
monitor reset halt
set mem inaccessible-by-default off
load
run
```

Segger J-Link Debugger

Connect the JLink Debugger to your UAVCAN device, and connect the JLink Debugger to your computer.

Power up your UAVCAN device, and run:

```
JLinkGDBServer -select USB=0 -device STM32F446RE -if SWD-DP -speed 20000 -vd
```

Open a second terminal, navigate to the directory that includes the `px4esc_1_6-bootloader.elf` for the esc and run:

```
arm-none-eabi-gdb px4esc_1_6-bootloader.elf
```

At the `gdb` prompt, run:

```
```tar ext :2331 load
```

```
Erasing Flash with SEGGER JLink Debugger

As a recovery method it may be useful to erase flash to factory defaults such that the fi
```

```
device erase ``
```

Replace `<name-of-device>` with the name of the microcontroller, e.g. STM32F446RE for the Pixhawk ESC 1.6 or STM32F302K8 for the SV2470VC ESC.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# UAVCAN 固件升级

官网英文原文地址：<http://dev.px4.io/uavcan-node-firmware.html>

## Vectorcontrol ESC Codebase (Pixhawk ESC 1.6 and S2740VC)

Download the ESC code:

```
git clone https://github.com/thiemar/vectorcontrol
cd vectorcontrol
```

### Flashing the UAVCAN Bootloader

Before updating firmware via UAVCAN, the Pixhawk ESC 1.6 requires the UAVCAN bootloader be flashed. To build the bootloader, run:

```
make clean && BOARD=px4esc_1_6 make -j8
```

After building, the bootloader image is located at `firmware/px4esc_1_6-bootloader.bin`, and the OpenOCD configuration is located at `openocd_px4esc_1_6.cfg`. Follow [these instructions](#) to install the bootloader on the ESC.

### Compiling the Main Binary

```
BOARD=s2740vc_1_0 make && BOARD=px4esc_1_6 make
```

This will build the UAVCAN node firmware for both supported ESCs. The firmware images will be located at `com.thiemar.s2740vc-v1-1.0-1.0.<git hash>.bin` and `org.pixhawk.px4esc-v1-1.6-1.0.<git hash>.binn`.

## Sapog Codebase (Pixhawk ESC 1.4)

Download the Sapog codebase:

```
git clone https://github.com/PX4/sapog
cd sapog
git submodule update --init --recursive
```

## Flashing the UAVCAN Bootloader

Before updating firmware via UAVCAN, the Pixhawk ESC 1.4 requires the UAVCAN bootloader be flashed. The bootloader can be built as follows:

```
cd bootloader
make clean && make -j8
cd ..
```

The bootloader image is located at `bootloader/firmware/bootloader.bin`, and the OpenOCD configuration is located at `openocd.cfg`. Follow [these instructions](#) to install the bootloader on the ESC.

## Compiling the Main Binary

```
cd firmware
make sapog.image
```

The firmware image will be located at `firmware/build/org.pixhawk.sapog-v1-1.0.<xxxxxxxx>.bin`, where `<xxxxxxxx>` is an arbitrary sequence of numbers and letters.

## Zubax GNSS

Please refer to the [project page](#) to learn how to build and flash the firmware. Zubax GNSS comes with a UAVCAN-capable bootloader, so its firmware can be updated in a uniform fashion via UAVCAN as described below.

## Firmware Installation on the Autopilot

The UAVCAN node file names follow a naming convention which allows the Pixhawk to update all UAVCAN devices on the network, regardless of manufacturer. The firmware files generated in the steps above must therefore be copied to the correct locations on an SD card or the PX4 ROMFS in order for the devices to be updated.

The convention for firmware image names is:

```
<uavcan name>-<hw version major>.<hw version minor>-<sw version major>.<sw version minor>.
<version hash>.bin
```

e.g. com.thiemar.s2740vc-v1-1.0-1.0.68e34de6.bin

However, due to space/performance constraints (names may not exceed 28 characters), the UAVCAN firmware updater requires those filenames to be split and stored in a directory structure like the following:

```
/fs/microsd/fw/<node name>/<hw version major>.<hw version minor>/<hw name>-<sw version
major>.<sw version minor>.<git hash>.bin
```

e.g. s2740vc-v1-1.0.68e34de6.bin

The ROMFS-based updater follows that pattern, but prepends the file name with \_ so you add the firmware in:

```
/etc/uavcan/fw/<device name>/<hw version major>.<hw version minor>/_<hw name>-<sw version
major>.<sw version minor>.<git hash>.bin
```

## Placing the binaries in the PX4 ROMFS

The resulting final file locations are:

- S2740VC ESC: ROMFS/px4fmu\_common/uavcan/fw/com.thiemar.s2740vc-v1/1.0/\_s2740vc-v1-
 1.0.<git hash>.bin
- Pixhawk ESC 1.6: ROMFS/px4fmu\_common/uavcan/fw/org.pixhawk.px4esc-v1/1.6/\_px4esc-v1-
 1.6.<git hash>.bin
  - Pixhawk ESC 1.4: `ROMFS/px4fmu\_common/uavcan/fw/org.pixhawk.sapog-
 v1/1.4/\_sapog-v1-1.4..bin`
  - Zubax GNSS v1: ROMFS/px4fmu\_common/uavcan/fw/com.zubax.gnss/1.0/gnss-1.0.<git
 has>.bin
  - Zubax GNSS v2: ROMFS/px4fmu\_common/uavcan/fw/com.zubax.gnss/2.0/gnss-2.0.<git
 has>.bin

Note that the ROMFS/px4fmu\_common directory will be mounted to /etc on Pixhawk.

## Starting the Firmware Upgrade process

When using the [PX4 Flight Stack](./2\_Concepts/flight\_stack.md), enable UAVCAN in the 'Power Config' section and reboot the system before attempting an UAVCAN firmware upgrade.

Alternatively UAVCAN firmware upgrading can be started manually on NSH via:

```
uavcan start
uavcan start fw
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

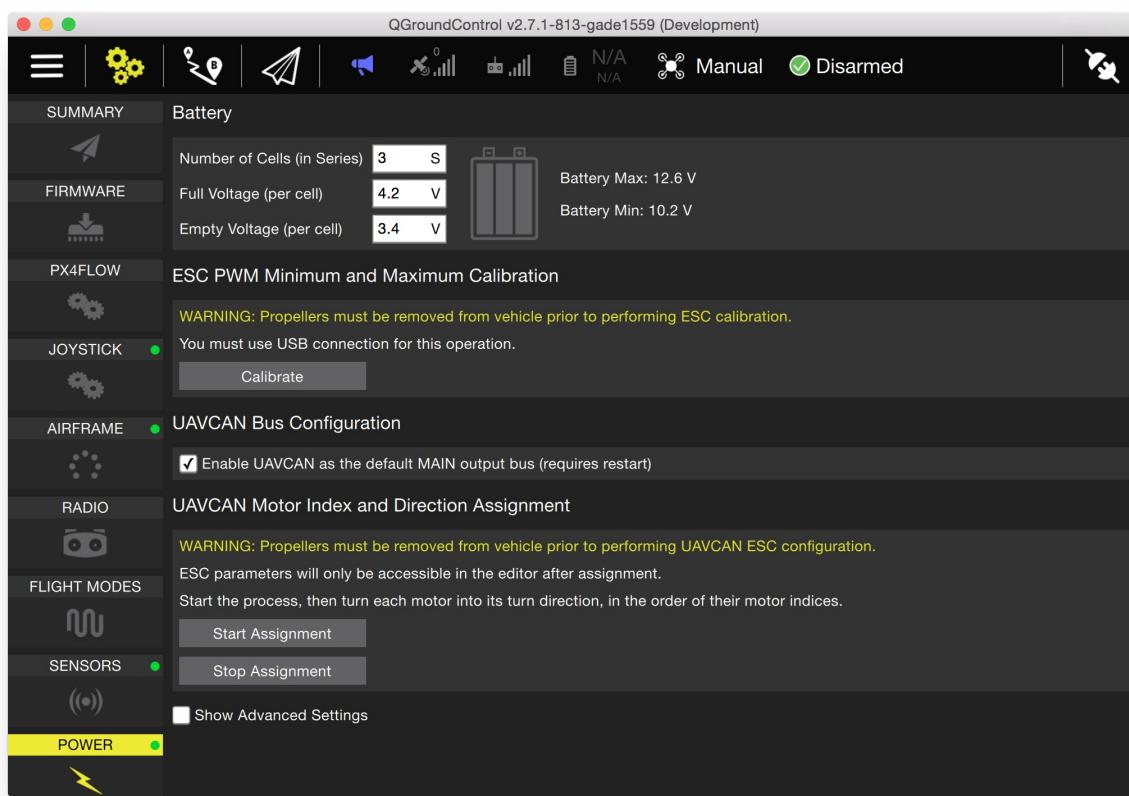
# UAVCAN Enumeration and Configuration

官网英文原文地址：<http://dev.px4.io/uavcan-node-enumeration.html>

Enable UAVCAN as the default motor output bus by ticking the 'Enable UAVCAN' checkbox as shown below. Alternatively the UAVCAN\_ENABLE parameter can be set to '3' in the QGroundControl parameter editor. Set it to '2' to enable CAN, but leave motor outputs on PWM.

Use [QGroundControl](#) and switch to the Setup view. Select the Power Configuration on the left. Click on the 'start assignment' button.

After the first beep, turn the propeller on the first ESC swiftly into the correct turn direction. The ESCs will all beep each time one is enumerated. Repeat this step for all motor controllers in the order as shown on the [motor map](#). This step has to be performed only once and does not need to be repeated after firmware upgrades.



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03



# PWM / GPIO

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# UART

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# uLanding Radar

官网英文原文地址：<http://dev.px4.io/uart-ulanding-radar.html>

The uLanding radar is a product from [Aerotenna](<http://aerotenna.com/sensors/>) and can be used to measure distance to an object.

## Enable the driver for your hardware

Currently, this radar device is supported by any hardware which runs the OS NuttX and which can offer a serial port for the

interface. Since flash space is small on some hardware you may have to enable building the driver for your target yourself.

To do so add the following line to the cmake config file which corresponds to the target you want to build for:

```
drivers/ulanding
```

All config files are located [here.]  
(<https://github.com/PX4/Firmware/tree/master/cmake/configs>)

## Start the driver

You will have to tell the system to start the driver for the radar during system startup.

You can simply add the following line to an [extras.txt](advanced-system-startup.md) file located on your SD card.

```
ulanding_radar start /dev/serial_port
```

In the upper command you will have to replace the last argument with the serial port you have connected the hardware to.

If you don't specify any port the driver will use /dev/ttyS2 which is the TELE2 port on Pixhawk.

### Warning

If you are connecting the radar device to TELE2 then make sure to set the parameter SYS\_COMPANION to 0. Otherwise the serial port

will be used by another application and you will get unexpected behaviour.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 调试以及高级主题

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 常见问题

官网英文原文地址：<http://dev.px4.io/advanced-faq.html>

## 编译错误

### 内存溢出

使用FMUv4架构可以获得双倍的内存。这一代的第一个可获得的板子是 [Pixracer](#)。

板子上可以装的代码量受限于它自身的内存大小。当添加额外的模块或者代码时很有可能超过内存容量。这将会导致“内存溢出”。上游的版本总是可以编译的，但是依赖于一个开发者添加了什么，它有可能在本地造成溢出。

```
region `flash' overflowed by 12456 bytes
```

为了改正这个问题，要么使用最近的硬件，要么移除对你来说不是必要的模块。配置在 [这里](#)。为了移除一个模块，可以直接注释掉它：

```
#drivers/trone
```

## USB错误

### 程序烧录一直不成功

在Ubuntu中，卸载调制解调器管理器：

```
sudo apt-get remove modemmanager
```

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22  
12:56:03

# PX4系统控制台

官网英文原文地址：<http://dev.px4.io/advanced-system-console.html>

该系统控制台（System Console）允许访问系统底层，调试输出和分析系统启动流程。访问系统控制台最方便的方式是使用Dronecode probe，但是也可以使用FTDI线（译者注：如果没有FTDI线，可用常见的USB转串口（TTL）模块代替，效果是一样的）。

## 系统控制台（System Console） vs. Shell

有多种shell，但只有一个控制台：系统控制台，它是打印所有引导输出（和引导中自动启动的应用程序）的位置。（可以理解为系统控制台是多个shell中唯一一个打印所有引导输出的shell）

- 系统控制台（第一shell）：硬件串口
- 其他shells: 连接到USB的Pixhawk(如Mac OS显示为 /dev/tty.usbmodem1)

USB shell:如果只是运行几个简单的命令或测试应用程序，连接到USB shell是足够的。

MAVLink可以在此使用，具体情况请查看下文。只有在需要开机调试或USB用于MAVlink连接地面站GCS的时候，才需要硬件串行控制台。

## Snapdragon Flight: Console接线

开发人员工具包里面有一个三个引脚接口板，它可以用于访问控制台。将捆绑的FTDI电缆连接到标头，并将接口板连接到扩展连接器。

## Pixracer / Pixhawk v3: Console接线

将6PJST SH 1 : 1线缆连接到Dronecode Probe，或者将连接线的每个接头按照如下所示连接到FTDI线上：

Pixracer / Pixhawk v3		FTDI	
1	+5V (red)		N/C
2	UART7 Tx	5	FTDI RX (黃)
3	UART7 Rx	4	FTDI TX (橙)
4	SWDIO		N/C
5	SWCLK		N/C
6	GND	1	FTDI GND (黑)

## Pixhawk v1: Console接线

系统控制台可以通过Dronecode Probe或FTDI线访问。这两个选项将在下面解释。

### 使用Dronecode Probe连接

将 Dronecode probe 的6P DF13 1:1线连接到Pixhawk的SERIAL4/5接口。

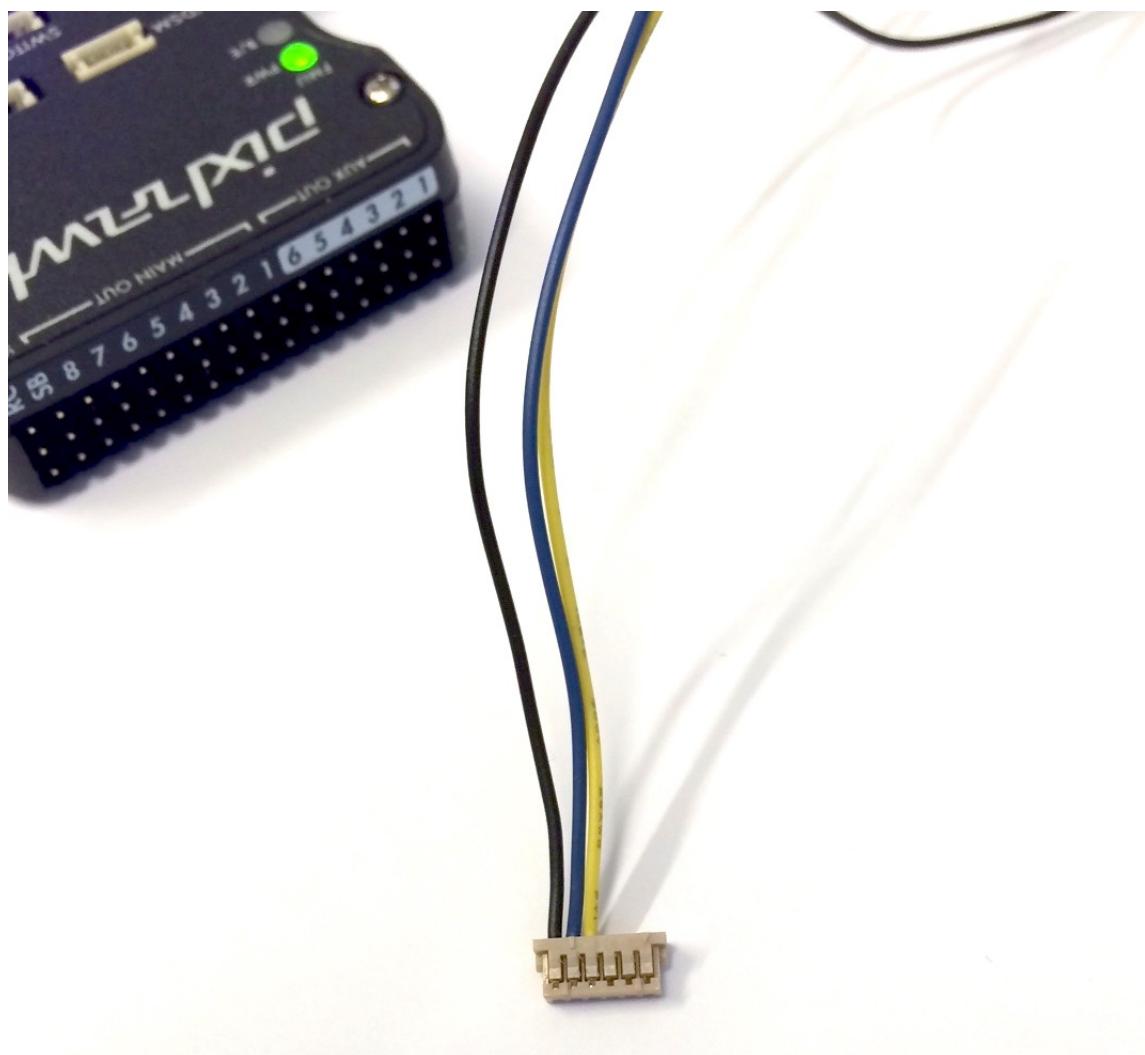


### 通过FTDI 3.3V 线（USB 转串口模块）连接

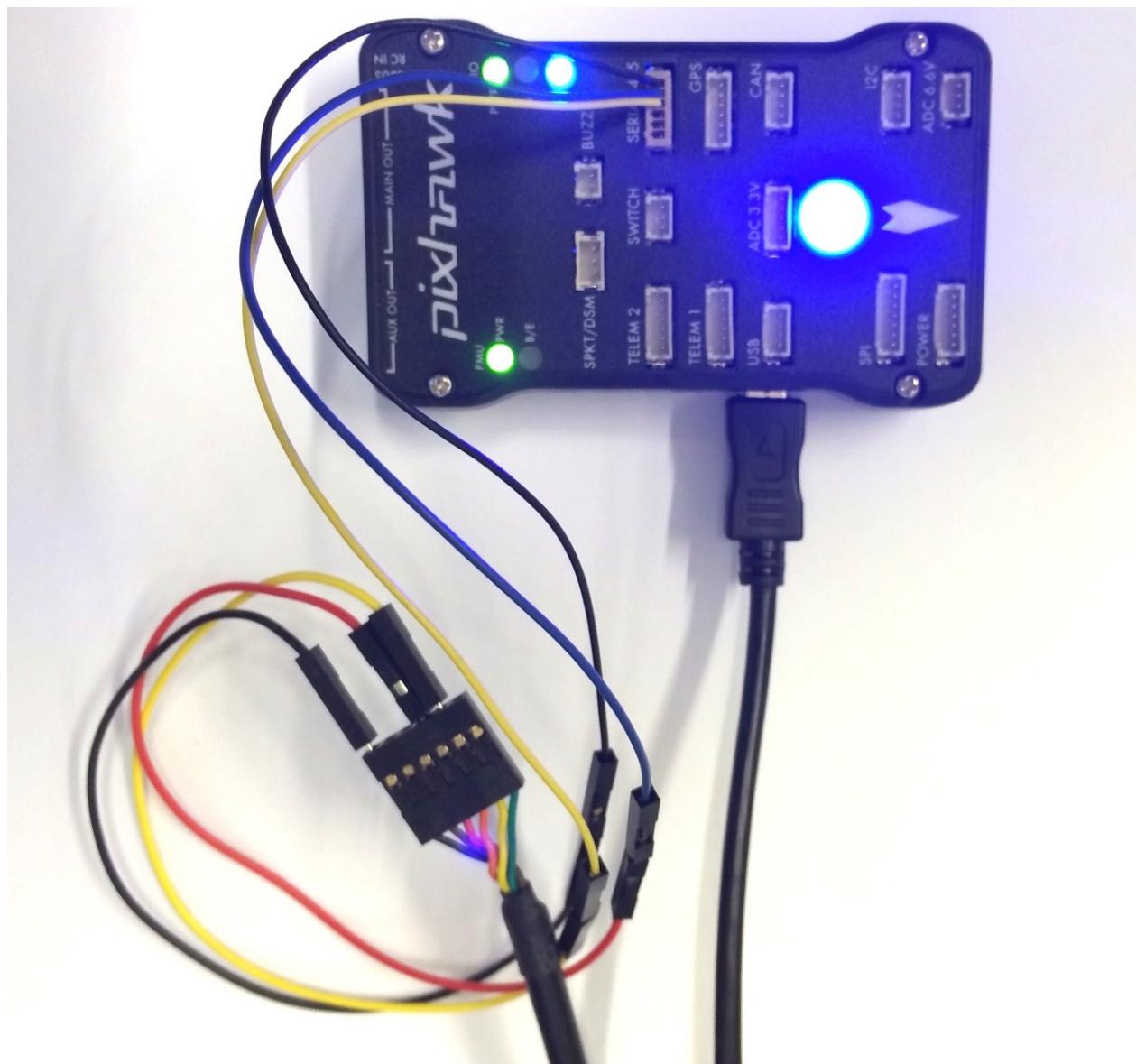
如果手头没有Dronecode Probe，也可以使用FTDI 3.3V (Digi-Key: [768-1015-ND](#))。

Pixhawk 1/2		FTDI	
1	+5V (红)		N/C
2	S4 Tx		N/C
3	S4 Rx		N/C
4	S5 Tx	5	FTDI RX (黄)
5	S5 Rx	4	FTDI TX (橙)
6	GND	1	FTDI GND (黑)

连接器引脚接线如图下图。



完整的布线如下。



## 打开控制台

控制台连接接线后，使用您选择的工具的默认串口或者以下描述的默认设置：（大部分新手读者看到这里，可能会困惑的是**console** 和 **screen** 的关系，不理解也没关系，不影响我们设置，仔细按照下面的教程进行设置，可以成功打开控制台）

### Linux / Mac OS: Screen

Ubuntu下安装screen (Mac OS 已经默认安装了):

```
sudo apt-get install screen
```

- 串行: Pixhawk v1 / Pixracer 使用 57600 波特率
- 串行: Snapdragon Flight 使用 115200 波特率

使用 `screen` 连接到正确的串口，配置为 BAUDRATE baud, 8 data bits, 1 stop bit（注：找到正确串口的方法如下：先在终端下输入 `ls /dev/tty*`，拔下串口设备（这里有一点需要注意，Pxhawk的小型USB口是一直插着给板子供电，拔的是上面的Dronecode probe）或者FTDI线（就是usb转串口线），然后重新输入 `ls /dev/tty*`，观察终端页面发生了什么样的变化，那个变化（少掉）的名称就是系统的串口）。在编者的机器上少了 `/dev/ttyUSB0`，说明串口设备的正确串口是 `/dev/ttUSB0`）。找到正确的串口后，重新连接串口设备。

常见名称，Linux下是 `/dev/ttUSB0` and `/dev/ttACM0`，Mac OS下是 `/dev/tt.usbserial-ABCBD`。（可以看到编者的linux系统的串口确实是常见名称 `/dev/ttUSB0`）

```
screen /dev/ttXXX BAUDRATE 8N1
```

注意上面的`/dve/ttXXX BAUDRATE 8N1`要替换为自己系统的正确串口和硬件波特率（如果是linux和pxhawk，加上编者上面的正确串口 `/dev/ttUSB0`，这条语句应该修改为 `screen /dev/ttUSB0 57600 8N1`）。正确输入上述命令后，终端会切换为**console**，如果没有切换为**console**或切换后输入没有反应，则插拔一下USB连接线或者串口连接线。重新输入 `screen /dev/ttUSB0 57600 8N1`，然后输入enter,出现nsh> 说明打开控制台成功。

## Windows: PuTTY

下载 [PuTTY](#) 并启动它。

然后选择“串行连接”，然后设置端口参数：

- 57600 baud
- 8 data bits
- 1 stop bit

## 控制台（Console）入门

键入`ls`查看本地文件系统，使用`free`查看剩余的可用RAM。当控制板单独供电时，控制台也将显示在系统引导日志。

```
nsh> ls
nsh> free
```

## MAVLink Shell

对于基于NuttX的系统（Pixhawk，Pixracer，...），NSH控制台也可以通过mavlink访问。它通过串行链路或WiFi（UDP / TCP）来工作。确保QGC没有运行，然后启动shell使用如下命令 `./Tools/mavlink_shell.py /dev/ttyACM0` （使用-h获得所有可用参数的描述）。

## Snapdragon DSP Console

当您通过USB连接到您的Snapdragon板，你可以访问POSIX上PX4的shell。与DSP侧（QuRT）的交互可以通过 `qshell` POSIX应用程序及其QuRT伴侣来开启。

通过USB连接的Snapdragon，打开mini-dm可以看到DSP的输出：

```
${HEXAGON_SDK_ROOT}/tools/mini-dm/Linux_Debug/mini-dm
```

运行主程序：

```
cd /home/linaro
./mainapp mainapp.config
```

您现在可以从linaro的shell以下语法来使用DSP加载的所有应用程序：

```
pxh> qshell command [args ...]
```

例如，要查看可用的QuRT应用程序：

```
pxh> qshell list_tasks
```

所执行的命令的输出显示在minidm。

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22  
12:56:03

# 系统启动

官网英文原文地址：<http://dev.px4.io/advanced-system-startup.html>

该PX4启动是由 `ROMFS/px4fmu_common/init.d` 文件夹下的 shell 脚本控制。

所有以数字和下划线（例如 `10000_airplane`）开头的文件的都是内置的机型配置。他们在编译时被导出成一个 `airframes.xml` 文件，它被 `QGroundControl` 解析后提供给机身选择界面使用。添加新的配置参照[此处](#)。

剩余的文件是常规启动逻辑的一部分，并且第一执行文件是 `rcS` 脚本，它调用所有其它的脚本。

## 调试系统启动

软件组件的驱动程序故障会导致启动中止。

一个不完整的启动往往表现为地面站中参数丢失，因为无法启动的应用程序没有初始化它们的参数。

调试启动序列正确的方法是连接[系统控制台](#) 和为电路板供电。由此产生的启动日志包含引导序列的详细信息，而且应该包含引导中止得原因。

## 常见启动故障的原因

- 一个必须的传感器发生故障
- 对于自定义的应用程序：该系统内存不足。运行 `free` 命令来查看可用内存量。
  - 软件故障或断言导致堆栈跟踪

## 更换系统启动

在大多数情况下，自定义修改默认的启动是更好的方法，这在后面介绍。如果要替换完整的 `boot`，创建一个 `/fs/microsd/etc/rc.txt` 文件，它在 microSD 卡上的 `etc` 文件夹中。如果这个文件存在，系统中没有什么会自动启动。

## 自定义系统启动

自定义系统启动的最好的方式是引进新的 [机身配置](#)。如果仅仅进行微调（如多启动一个应用程序或只是使用不同的混合器），启动时可以使用特别的 `hook`。

该系统启动文件是UNIX文件，这需要UNIX形式的行结尾。如果在Windows上编辑，请使用合适的编辑器。

主要有三个hook。需要注意的是microSD卡的根目录文件夹路径是 /fs/microsd。

- /fs/microsd/etc/config.txt
- /fs/microsd/etc/extras.txt
  - /fs/microsd/mixers/NAME\_OF\_MIXER

## 自定义配置 (config.txt)

主系统配置完成后，且在启动前，加载 config.txt 文件，此时允许修改shell变量。

## 启动其他应用程序

extras.txt 可用于在主系统引导后启动额外的应用程序。通常，这些将是载荷控制器或类似的可选自定义组件。

## 启动一个自定义的mixer

系统默认从 /etc/mixers 加载mixer。如果在 /fs/microsd/etc/mixers 有相同名称的文件中存在，那么将加载该文件来代替。这允许定制混合器文件，而不需要重新编译固件。

© PX4WIKI team all right reserved，powered by Gitbook 该文件修订时间：2017-01-22  
12:56:03

# 参数 & 配置

官网英文原文地址：<http://dev.px4.io/advanced-configurations.html>

PX4使用参数子系统（实际就是浮点和整型数据的列表）和文本文件（用来配置Mixer混合器和启动脚本）来储存相关配置。

关于系统启动 和机体参数配置 的实现在其他章节有详细讲述。这部分主要是详细讨论参数子系统。

## 命令行的使用

PX4系统控制台 提供了 `param` 命令，可以对参数进行设置、访问、保存，以及从文件中导入和保存到文件。

### 访问和设置参数

命令行`param show` 可以列出所有系统参数：

```
param show
```

参数名+字符可以选择对应的参数进行操作：

```
nsh> param show RC_MAP_A*
Symbols: x = used, + = saved, * = unsaved
x RC_MAP_AUX1 [359,498] : 0
x RC_MAP_AUX2 [360,499] : 0
x RC_MAP_AUX3 [361,500] : 0
x RC_MAP_ACRO_SW [375,514] : 0

723 parameters total, 532 used.
```

### 导出和加载参数

一般的保存命令可以保存参数到默认的文件中：

```
param save
```

如果保存后面加上路径，将会保存参数到新的位置

```
param save /fs/microsd/vtol_param_backup
```

加载参数有两种方法: `param load` 加载文件并用文件中的数据代替现有参数设置, 最终把以前某个状态储存的数据一一复制过来 `param import` 这个命令更为精妙, 它只改变与默认设置不同的参数。这个命令有重要的作用, 比如在进行最初校准但不进行其他配置时, 导入之前校准的参数就可以只改变校准数据而不对其他配置操作。

覆盖现有参数:

```
param load /fs/microsd/vtol_param_backup
```

合并现有参数和储存的参数 (储存文件中与默认参数不同的参数覆盖默认参数):

```
param import /fs/microsd/vtol_param_backup
```

## C / C++ API

PX4还有独立的C和C++接口访问配置数据。

Discuss param C / C++ API.

```
int32_t param = 0;
param_get(param_find("PARAM_NAME"), ¶m);
```

## 参数数据元

PX4使用一个参数数据元系统把参数展示给用户。正确的合适的数据元对地面站的用户体验有重要意义。

一段传统的数据元如下所示：

```
/**
 * Pitch P gain
 *
 * Pitch proportional gain, i.e. desired angular speed in rad/s for error 1 rad.
 *
 * @unit 1/s
 * @min 0.0
 * @max 10
 * @decimal 2
 * @increment 0.0005
 * @reboot_required true
 * @group Multicopter Attitude Control
 */
PARAM_DEFINE_FLOAT(MC_PITCH_P, 6.5f);
```

各行的作用：

```
/**
 * <title>
 *
 * <longer description, can be multi-line>
 *
 * @unit <the unit, e.g. m for meters>
 * @min <the minimum sane value. Can be overridden by the user>
 * @max <the maximum sane value. Can be overridden by the user>
 * @decimal <the minimum sane value. Can be overridden by the user>
 * @increment <the "ticks" in which this value will increment in the UI>
 * @group <a title for parameters which form a group>
 */
PARAM_DEFINE_FLOAT(MC_PITCH_P, 6.5f);
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22  
12:56:03

# 嵌入式调试

官网英文原文地址：<http://dev.px4.io/advanced-gdb-debugging.html>

The autopilots running PX4 support debugging via GDB or LLDB.

## Identifying large memory consumers

The command below will list the largest static allocations:

```
arm-none-eabi-nm --size-sort --print-size --radix=dec build_px4fmu-v2_default/src/firmware
```

This NSH command provides the remaining free memory:

```
free
```

And the top command shows the stack usage per application:

```
top
```

Stack usage is calculated with stack coloring and thus is not the current usage, but the maximum since the start of the task.

## Heap allocations

Dynamic heap allocations can be traced on POSIX in SITL with [gperftools](#). Once installed, it can be used with:

- Run jmavsim: `./Tools/jmavsim_run.sh`
- Then:

```
cd build_posix_sitl_default/tmp
export HEAPPROFILE=/tmp/heaprofile.hprof
env LD_PRELOAD=/lib64/libtcmalloc.so ./src/firmware posix/px4 posix-configs/SITL/init/lp
pprof --pdf ./src/firmware posix/px4 /tmp/heaprofile.hprof.0001.heap > heap.pdf
```

It will generate a pdf with a graph of the heap allocations. The numbers in the graph will all be zero, because they are in MB. Just look at the percentages instead. They show the live memory (of the node and the subtree), meaning the memory that was still in use at the end.

If it does not generate heap dumps while running the `px4` app you might need to change the settings of the profiler. On some systems it is necessary to set an interval time when to write the dumps:

```
Specify interval in seconds
export HEAP_PROFILE_TIME_INTERVAL=10
```

See the [gperftools docs](#) for more information.

## Sending MAVLink debug key / value pairs

The code for this tutorial is available here:

- [Debug Tutorial Code](#)
- [Enable the tutorial app](#) by uncommenting / enabling the mavlink debug app in the config of your board

All required to set up a debug publication is this code snippet. First add the header file:

```
#include <uORB/uORB.h>
#include <uORB/topics/debug_key_value.h>
```

Then advertise the debug value topic (one advertisement for different published names is sufficient). Put this in front of your main loop:

## Debugging Hard Faults in NuttX

A hard fault is a state when the operating system detects that it has no valid instructions to execute. This is typically the case when key areas in RAM have been corrupted. A typical scenario is when incorrect memory access smashed the stack and the processor sees that the address in memory is not a valid address for the microprocessors's RAM.

- NuttX maintains two stacks: The IRQ stack for interrupt processing and the user stack
- The stack grows downward. So the highest address in the example below is 0x20021060, the size is 0x11f4 (4596 bytes) and consequently the lowest address is 0x2001fe6c.

```

Assertion failed at file:armv7-m/up_hardfault.c line: 184 task: ekf_att_pos_estimator
sp: 20003f90
IRQ stack:
 base: 20003fdc
 size: 000002e8
20003f80: 080d27c6 20003f90 20021060 0809b8d5 080d288c 000000b8 08097155 00000010
20003fa0: 20003ce0 00000003 00000000 0809bb61 0809bb4d 080a6857 e000ed24 080a3879
20003fc0: 00000000 2001f578 080ca038 000182b8 20017cc0 0809bad1 20020c14 00000000
sp: 20020ce8
User stack:
 base: 20021060
 size: 000011f4
20020ce0: 60000010 2001f578 2001f578 080ca038 000182b8 0808439f 2001fb88 20020d4c
20020d00: 20020d44 080a1073 666b655b 65686320 205d6b63 6f6c6576 79746963 76696420
20020d20: 65747265 63202c64 6b636568 63636120 63206c65 69666e6f 08020067 0805c4eb
20020d40: 080ca9d4 0805c21b 080ca1cc 080ca9d4 385833fb 38217db9 00000000 080ca964
20020d60: 080ca980 080ca9a0 080ca9bc 080ca9d4 080ca9fc 080caa14 20022824 00000002
20020d80: 2002218c 0806a30f 08069ab2 81000000 3f7ffffec 00000000 3b4ae00c 3b12eaa6
20020da0: 00000000 00000000 080ca010 4281fb70 20020f78 20017cc0 20020f98 20017cdc
20020dc0: 2001ee0c 0808d7ff 080ca010 00000000 3f800000 00000000 080ca020 3aa35c4e
20020de0: 3834d331 00000000 01010101 00000000 01010001 000d4f89 000d4f89 000f9fda
20020e00: 3f7d8df4 3bac67ea 3ca594e6 be0b9299 40b643aa 41ebe4ed bcc04e1b 43e89c96
20020e20: 448f3bc9 c3c50317 b4c8d827 362d3366 b49d74cf ba966159 00000000 00000000
20020e40: 3eb4da7b 3b96b9b7 3eedad6a 00000000 00000000 00000000 00000000 00000000
20020e60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
20020e80: 00000016 00000000 00000000 00010000 00000000 3c23d70a 00000000 00000000
20020ea0: 00000000 20020f78 00000000 2001ed20 20020fa4 2001f498 2001f1a8 2001f500
20020ec0: 2001f520 00000003 2001f170 ffffffe9 3b831ad2 3c23d70a 00000000 00000000
20020ee0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
20020f00: 00000000 00000000 00000000 00000000 2001f4f0 2001f4a0 3d093964 00000001
20020f20: 00000000 0808ae91 20012d10 2001da40 0000260b 2001f577 2001da40 0000260b
20020f40: 2001f1a8 08087fd7 08087f9d 080cf448 0000260b 080afab1 080afa9d 00000003
20020f60: 2001f577 0809c577 2001ed20 2001f4d8 2001f498 0805e077 2001f568 20024540
20020f80: 00000000 00000000 00000000 0000260b 3d093a57 00000000 2001f540 2001f4f0
20020fa0: 0000260b 3ea5b000 3ddbf5fa 00000000 3c23d70a 00000000 00000000 000f423f
20020fc0: 00000000 000182b8 20017cc0 2001ed20 2001f4e8 00000000 2001f120 0805ea0d
20020fe0: 2001f090 2001f120 2001eda8 ffffffff 000182b8 00000000 00000000 00000000
20021000: 00000000 00000000 00000009 00000000 08090001 2001f93c 0000000c 00000000
20021020: 00000101 2001f96c 00000000 00000000 00000000 00000000 00000000 00000000
20021040: 00000000 00000000 00000000 00000000 00000000 0809866d 00000000 00000000
R0: 20000f48 0a91ae0c 20020d00 20020d00 2001f578 080ca038 000182b8 20017cc0
R8: 2001ed20 2001f4e8 2001ed20 00000005 20020d20 20020ce8 0808439f 08087c4e
xPSR: 61000000 BASEPRI: 00000000 CONTROL: 00000000
EXC_RETURN: ffffffe9

```

To decode the hardfault, load the `exact` binary into the debugger:

```
arm-none-eabi-gdb build_px4fmu-v2_default/src/firmware/nuttx/firmware_nuttx
```

Then in the GDB prompt, start with the last instructions in R8, with the first address in flash (recognizable because it starts with `0x080`, the first is `0x0808439f`). The execution is left to right. So one of the last steps before the hard fault was when `mavlink_log.c` tried to publish something,

```
(gdb) info line *0x0808439f
Line 77 of ".../src/modules/systemlib/mavlink_log.c" starts at address 0x8084398
and ends at 0x80843a0 .
```

```
(gdb) info line *0x08087c4e
Line 311 of ".../src/modules/uORB/uORBDevices_nuttx.cpp"
starts at address 0x8087c4e
and ends at 0x8087c52 .
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 仿真调试

官网英文原文地址：<http://dev.px4.io/simulation-debugging.html>

As the simulation is running on the host machine, all the desktop development tools are available.

## CLANG Address Sanitizer (Mac OS, Linux)

The Clang address sanitizer can help to find alignment (bus) errors and other memory faults like segmentation faults. The command below sets the right compile options.

```
shmake clean # only required on first address sanitizer run after a normal
buildMEMORY_DEBUG=1 make posix jmavsim
```

## Valgrind

```
shbrew install valgrind
```

or

```
shsudo apt-get install valgrind
```

Add instructions how to run Valgrind

## Start combinations

SITL can be launched with and without debugger attached and with either jMAVSIM or Gazebo as simulation backend. This results in the start options below:

```
make posix_sitl_default jmavsimmake posix_sitl_default jmavsim__gdbmake posix_sitl_defau
make posix_sitl_default gazebomake posix_sitl_default gazebo__gdbmake posix_sitl_default
make posix_sitl_lpe jmavsimmake posix_sitl_lpe jmavsim__gdbmake posix_sitl_lpe jmavsim__
make posix_sitl_lpe gazebomake posix_sitl_lpe gazebo__gdbmake posix_sitl_lpe gazebo__ll
```

where the last parameter is the <viewer\_model\_debugger> triplet (using three underscores implies the default 'iris' model). This will start the debugger and launch the SITL application. In order to break into the debugger shell and halt the execution, hit `CTRL-C` :

```
Process 16529 stopped* thread #1: tid = 0x114e6d, 0x00007fff90f4430a libsystem_kernel.dylib
```

In order to not have the DriverFrameworks scheduling interfere with the debugging session `SIGCONT` should be masked in LLDB and GDB:

```
(lldb) process handle SIGCONT -n false -p false -s false
```

Or in the case of GDB:

```
(gdb) handle SIGCONT noprint nostop
```

After that the The lldb or gdb shells behave like normal sessions, please refer to the LLDB / GDB documentation.

The last parameter, the <viewer\_model\_debugger> triplet, is actually passed to make in the build directory, so

```
make posix_sitl_lpe jmavsim_gdb
```

is equivalent with

```
make posix_sitl_lpe # Configure with cmake
make -C build_posix_sitl_lpe jmavsim_gdb
```

A full list of the available make targets in the build directory canbe obtained with:

```
make help
```

but for your convenience, a list with just the <viewer\_model\_debugger> tripletsis printed with the command

```
make list_vmd_make_targets
```

## Compiler optimization

It is possible to suppress compiler optimization for given executables and/or modules (as added by cmake with `add_executable` or `add_library`) when configuring for `posix_sitl_*`. This can be handy when it is necessary to step through code with a debugger or print variables that would otherwise be optimized out.

To do so, set the environment variable `PX4_NO_OPTIMIZATION` to be a semi-colon-separated list of regular expressions that match the targets that need to be compiled without optimization. This environment variable is ignored when the configuration isn't `posix_sitl_*`.

For example,

```
export PX4_NO_OPTIMIZATION='px4;^modules__uORB;^modules__systemlib$'
```

would suppress optimization of the targets: `platforms__posix_px4_layer`, `modules__systemlib`, `modules__uORB`, `examples__px4_simple_app`, `modules__uORB_uORB_tests` and `px4`.

The targets that can be matched with these regular expressions can be printed with the command:

```
make -C build_posix_sitl_* list_cmake_targets
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# Send Debug String / Float Pairs

官网英文原文地址：<http://dev.px4.io/advanced-debug-values.html>

It is often necessary during software development to output individual important numbers.

This is where the generic 'NAMED\_VALUE' packets of MAVLink come in.

## Files

The code for this tutorial is available here:

- Debug Tutorial Code
- Enable the tutorial app by uncommenting / enabling the mavlink debug app in the config of your board

All required to set up a debug publication is this code snippet. First add the header file:

```
```#include
```

```
#include <uORB/topics/debug_key_value.h>```
```

Then advertise the debug value topic (one advertisement for different published names is sufficient). Put this in front of your main loop:

```
```/* advertise debug value */
struct debug_key_value_s dbg = { .key = "velx", .value = 0.0f };
orb_advert_t pub_dbg = orb_advertise(ORB_ID(debug_key_value), &dbg);```
```

And sending in the main loop is even simpler:

```
dbg.value = position[0]; orb_publish(ORB_ID(debug_key_value), pub_dbg, &dbg);
```

Multiple debug messages must have enough time between their respective publishings for Mavlink to process them. This means that either the code must wait between publishing multiple debug messages, or alternate the messages on each function call iteration.

The result in QGroundControl then looks like this on the real-time plot:



© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# FAKE GPS

官网英文原文地址：<http://dev.px4.io/advanced-fake-gps.html>

This page shows you how to use mocap data to fake gps.

The setup looks as follows: There is a "VICON computer" which has the required software installed + [ROS](#) + [vicon\\_bridge](#) and sends this data over network to "your Computer". On "your computer" you should have ROS + MAVROS installed. In MAVROS there is a script which simulates gps data out of mocap data. "Your computer" then sends the data over 3DR radiometry to the pixhawk. *NOTE:* The "VICON computer" and "your computer" can be the same, of course.

## Prerequisites

- MOCAP system (in this example, VICON is used)
- Computer with ROS, MAVROS and Vicon\_bridge
- 3DR radiometry set

## Procedure

### Step 1

Make sure "your computer" is in the same network as the "VICON computer" (maybe you need a wireless adapter). Create two files on the "VICON computer": "launch\_fake\_gps.sh" and "launch\_fake\_gps\_distorted.sh"

Add the following two lines to the "launch\_fake\_gps.sh" file and replace xxx.xxx.x.xxx with the IP address of "your computer" (you can get the IP address by typing "ifconfig" in the terminal).

```
export ROS_MASTER_URI=http://xxx.xxx.x.xxx:11311
roslaunch vicon_bridge vicon.launch $@
```

Next, add the following two lines to the "launch\_fake\_gps\_distorted.sh" file and replace xxx.xxx.x.xxx with the IP address of "your computer".

```
export ROS_MASTER_URI=http://xxx.xxx.x.xxx:11311
rosrun vicon_bridge tf_distort $@
```

Put markers on your drone and create a model in the MOCAP system (later referred to as `yourModelName`).

## Step 2

Run

```
$ roscore
```

on "your computer".

## Step 3

Run

```
$ sh launch_fake_gps.sh
```

and

```
$ sh launch_fake_gps_distorted.sh
```

in two different terminals on the "VICON computer" in the directory where you created the two files.

## Step 4

On "your computer" run

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

A new window should open where you can select "tf\_distort". With this tool you can edit the parameters to distort the MOCAP data.

To simulate GPS we use:

- publish rate = 5.0Hz
- tf\_frame\_in = vicon/`yourModelName`/`yourModelName` (e.g. vicon/DJI\_450/DJI\_450)
- delay = 200ms

- $\sigma_{xy} = 0.05m$
- $\sigma_z = 0.05m$

## Step 5

Connect your pixhawk in QGroundControl. Go to PARAMETERS -> System and change SYS\_COMPANION to 257600 (enables magic mode;)).

Next, go to PARAMETERS -> MAVLink and change MAV\_USEHILGPS to 1 (enable HIL GPS).

Now, go to PARAMETERS -> Attitude Q estimator and change ATT\_EXT\_HDG\_M to 2 (use heading from motion capture).

Last but not least, go to PARAMETERS -> Position Estimator INAV and change INAV\_DISAB\_MOCAP to 1 (disable mocap estimation).

*NOTE:* if you can't find the above stated parameters, check PARAMETERS -> default Group

## Step 6

Next, open "mocap\_fake\_gps.cpp". You should find it at:

yourCatkinWS/src/mavros/mavros\_extras/src/plugins/mocap\_fake\_gps.cpp

Replace DJI\_450/DJI\_450 in

```
mocap_tf_sub = mp_nh.subscribe("/vicon/DJI_450/DJI_450_drop", 1, &MocapFakeGPSPlugin::moc
```

with your model name (e.g. /vicon/yourModelName/yourModelname\_drop). The "\_drop" will be explained in the next step.

## Step 7

In step 5, we enabled heading from motion capture. Therefore pixhawk does not use the original North, East direction, but the one from the motion capture system. Because the 3DR radiometry device is not fast enough, we have to limit the rate of our MOCAP data. To do this run

```
$ rosrun topic_tools drop /vicon/yourModelName/yourModelName 9 10
```

This means, that from the rostopic `/vicon/yourModelName/yourModelName`, 9 out of 10 messages will be dropped and published under the topic name `"/vicon/yourModelName/yourModelName_drop"`.

## Step 8

Connect the 3DR radiometry with the pixhawk TELE2 and the counter part with your computer (USB).

## Step 9

Go to your catkinWS and run

```
$ catkin build
```

and afterwards

```
$ rosrun mavros px4.launch fcu_url:=/dev/ttyUSB0:57600
```

That's it! Your pixhawk now gets GPS data and the light should pulse in green color.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 相机触发器

官网英文原文地址:<http://dev.px4.io/advanced-camera-trigger.html>

相机触发驱动器是为了让AUX端口发出一个脉冲来触发相机。这个可以用于多个应用程序，包括航测和重建照片、同步多相机系统或者视觉惯性导航。

除了会发送一个脉冲，mavlink会传回一个序列号（当前所拍摄的图像的一个序列号）和拍摄的时间。支持三种不同的模式：

- 触发 1 就像一个基本的定时器，可以通过系统控制台分别设置启用和禁用 相机触发使能 或 相机触发不使能。可以重复设置间隔时间，来执行相机触发。
- 触发模式 2 用一个开关来定时何时曝光。
- 触发模式 3 基于远程触发。每次超过设定的水平距离时都要触发拍摄。然而，两个相机之间的最短时间间隔由设定的触发间隔时间决定。

在 触发模式 0 触发关闭。

想找到与相机触发模块有关的参数配置的完整列表，请参考 [参考](#) 页。

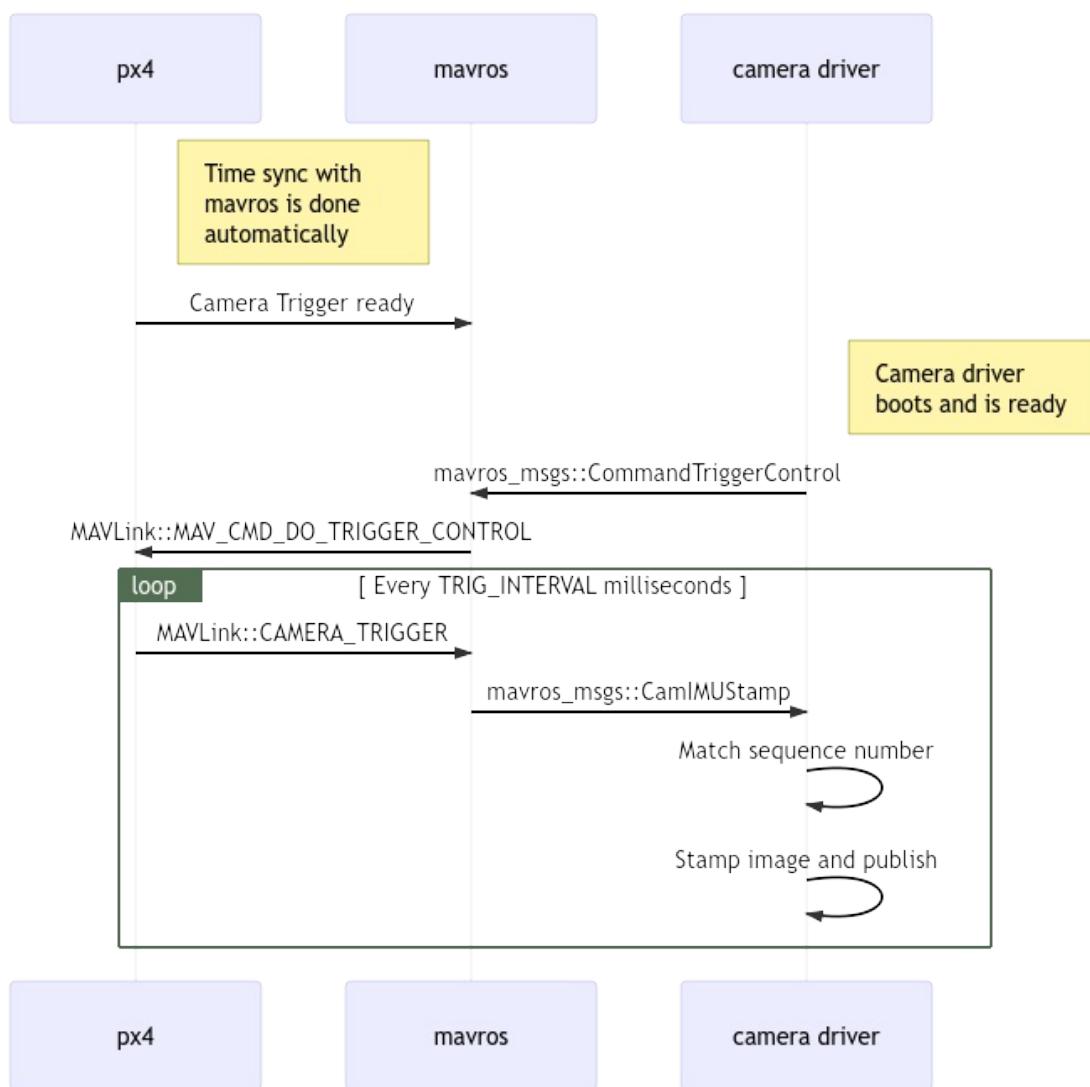
如果这是你第一次启用相机触发应用程序，记得要修改 触发模式 参数为 1, 2 或 3。

## 相机IMU设置同步的例程

在这个例程中，我们将对IMU测量同步可视化数据的基础来建立一个立体视觉惯性导航系统(VINS)。很明显，这里的想法是IMU不参与测量，而是当到达设置的时间点时触发拍照，为什么提供准确的数据的迭代算法。

自驾仪和外设有不同的时钟(自驾仪的启动时间和UNIX外设的启动时间)，所以他们有不同的时间，我们直接看时钟之间的时间偏移量。这个偏移量是添加或从中减去mavlink消息的时间（如高分辨率的IMU）（如该译者在PX4同伴和mavlink的接收端）。实际的同步算法是网络时间协议(NTP)算法的修改版本，使用指数移动平均平滑跟踪时间偏移。如果使用高宽带的车载连接会自动完成这种同步连接。要获取同步的图像数据和惯性测量数据，我们就要把相机的触发信号输入引脚连接到自驾仪的GPIO引脚。从中获取惯性测量的时间点数据和图像序列号记录并发送到配套的计算机（相机触发信息），缓存这些数据和从相机获得的图像。这些是匹配了序列号、时间点数据的图像。

下面的图表说明了事件的发生，必须以正确的时间点匹配我们的图像序列号。



## 步骤 1

首先, 设置触发模式为模式1来使驱动程序等待开始命令并启动你的FCU来获取其余的参数.

## 步骤 2

对于这个例程的目的, 我们将配置触发器操作一个Point Grey Firefly MV cam运行在30FPS.

- 触发间隔时间: 33.33 ms ()
- 触发信号极性: 0, 拉低电平
- 触发有效时间: 0.5 ms, 无. 手动设置触发信号仅至少需要1微妙时间.
- 触发模式: 1, 因为我们希望我们的相机驱动程序准备好接收图像再开始触发. 这是正确处理序列号数据的必要条件和基础.
- 触发引脚: 12, 无.

## 步骤3

将你的相机连接到AUX接口的信号和地线.

## 步骤 4

你必须按照上面的顺序来修改你的驱动程序. 公开的实现方法请参考[IDS 成像 UEye 可兼容的相机列表](#)请参考[兼容IEEE1394](#).

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# Logging

官网英文原文地址：<http://dev.px4.io/advanced-logging.html>

## Logging

This describes the new logger, `SYS_LOGGER` set to 1.

The logger is able to log any ORB topic with all included fields. Everything necessary is generated from the `.msg` file, so that only the topic name needs to be specified. An optional interval parameter specifies the maximum logging rate of a certain topic. All existing instances of a topic are logged.

The output log format is [ULog](#).

## Usage

By default, logging is automatically started when arming, and stopped when disarming. A new log file is created for each arming session on the SD card. To display the current state, use `logger status` on the console. If you want to start logging immediately, use `logger on`. This overrides the arming state, as if the system was armed. `logger off` undoes this.

Use

```
logger help
```

for a list of all supported logger commands and parameters.

## Configuration

The list of logged topics can be customized with a file on the SD card. Create a file `etc/logging/logger_topics.txt` on the card with a list of topics:

```
<topic_name>, <interval>
```

The `<interval>` is optional, and if specified, defines the minimum interval in ms between two logged messages of this topic. If not specified, the topic is logged at full rate.

The topics in this file replace all of the default logged topics.

## Scripts

There are several scripts to analyze and convert logging files in the [pyulog](#) repository.

## Dropouts

Logging dropouts are undesired and there are a few factors that influence the amount of dropouts:

- Most SD cards we tested exhibit multiple pauses per minute. This shows itself as a several 100 ms delay during a write command. It causes a dropout if the write buffer fills up during this time. This effect depends on the SD card (see below).
- Formatting an SD card can help to prevent dropouts.
- Increasing the log buffer helps.
- Decrease the logging rate of selected topics or remove unneeded topics from being logged (`info.py <file>` is useful for this).

## SD Cards

The following provides performance results for different SD cards. Tests were done on a Pixracer; the results are applicable to Pixhawk as well.

SD Card	Mean Seq. Write Speed [KB/s]	Max Write Time / Block (average) [ms]
SanDisk Extreme U3 32GB	461	<b>15</b>
Sandisk Ultra Class 10 8GB	348	40
Sandisk Class 4 8GB	212	60
SanDisk Class 10 32 GB (High Endurance Video Monitoring Card)	331	220
Lexar U1 (Class 10), 16GB High-Performance	209	150
Sandisk Ultra PLUS Class 10 16GB	196	500
Sandisk Pixtor Class 10 16GB	334	250
Sandisk Extreme PLUS Class 10 32GB	332	150

More important than the mean write speed is the maximum write time per block (of 4 KB). This defines the minimum buffer size: the larger this maximum, the larger the log buffer needs to be to avoid dropouts. Logging bandwidth with the default topics is around 50 KB/s, which all of the SD cards satisfy.

By far the best card we know so far is the **SanDisk Extreme U3 32GB**. This card is recommended, because it does not exhibit write time spikes (and thus virtually no dropouts). Different card sizes might work equally well, but the performance is usually different.

You can test your own SD card with `sd_bench -r 50`, and report the results to <https://github.com/PX4/Firmware/issues/4634>.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 飞行日志分析

官网英文原文地址：[http://dev.px4.io/flight\\_log\\_analysis.html](http://dev.px4.io/flight_log_analysis.html)

这里有几个分析PX4飞行日志的软件，描述如下：

## Log Muncher

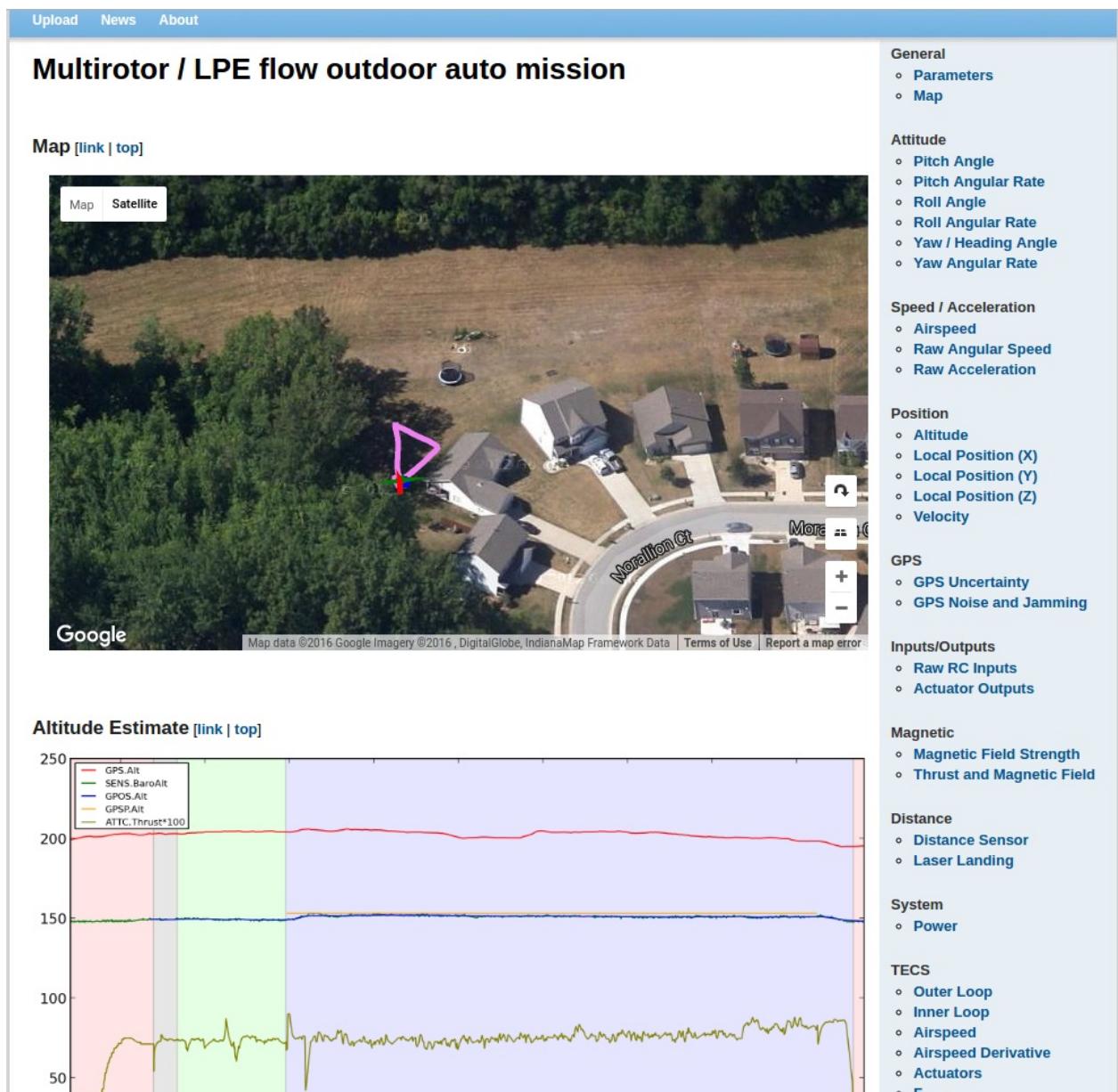
注意：Log Muncher只可以被用来查看先前 .px4log 格式的日志`

### 上传

用户可直接访问网站并直接上传log: <http://logs.uaventure.com>

The screenshot shows the Log Muncher website interface. At the top, there is a dark header bar with the text "LogMuncher [Flight Log Analyser]". Below it is a light blue navigation bar with links for "Upload", "News", and "About". The main area has a white background. On the left, there is a form titled "Upload" with fields for "Email:" (an empty input field), "Title:" (an empty input field), and a file upload section with a "Choose File" button and a message "No file chosen". Below these is a "Upload" button. On the right, there is a sidebar with a dark header "Important" containing a note about uploaded log files potentially containing location information and accepting that this information is stored on their system. At the bottom of the page, there is a dark footer bar with the text "Copyright © UAVenture AG".

### 结果



## Example Log

### 优点

- 基于网页，便于终端用户
- 用户可以上传日志并和别人分享

### 缺点

- 分析非常有限，没有定制功能

## Flight Review

Flight Review是Log Muncher的继任者，与新的ULog记录格式结合使用。

## 示例



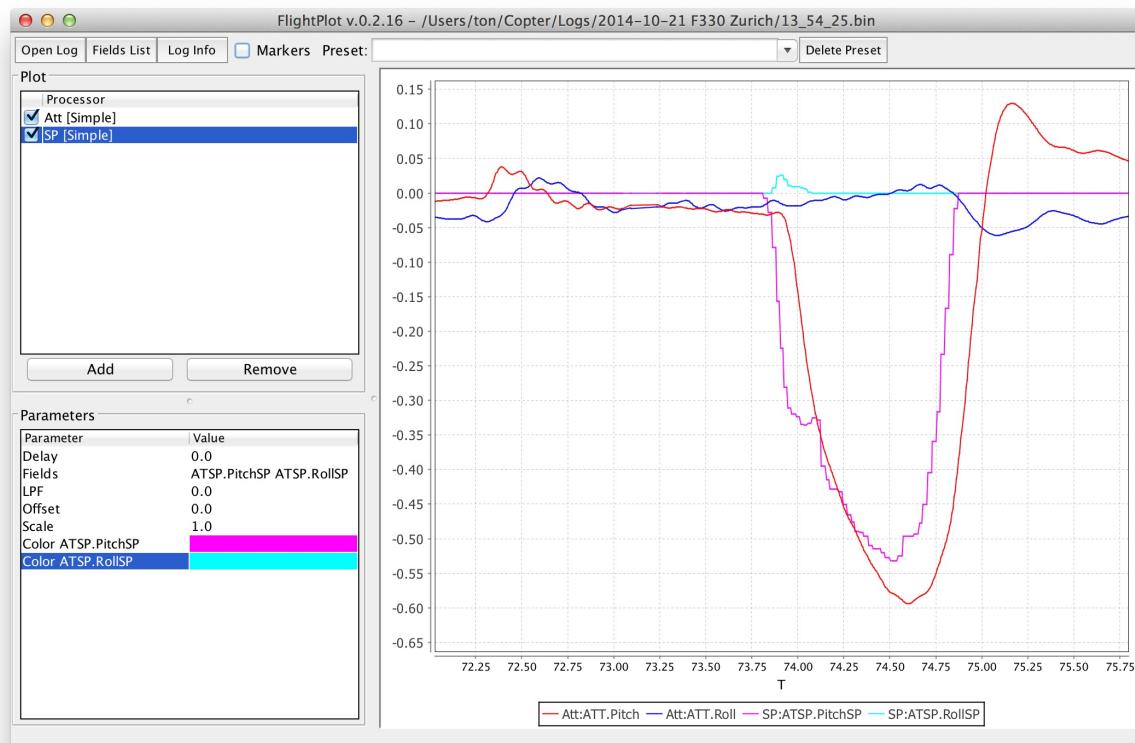
## 优点

- 基于网页，便于终端用户
- 用户可以上传并和别人分享
- 交互式的画图体验

## 缺点

- 没有定制功能

## FlightPlot



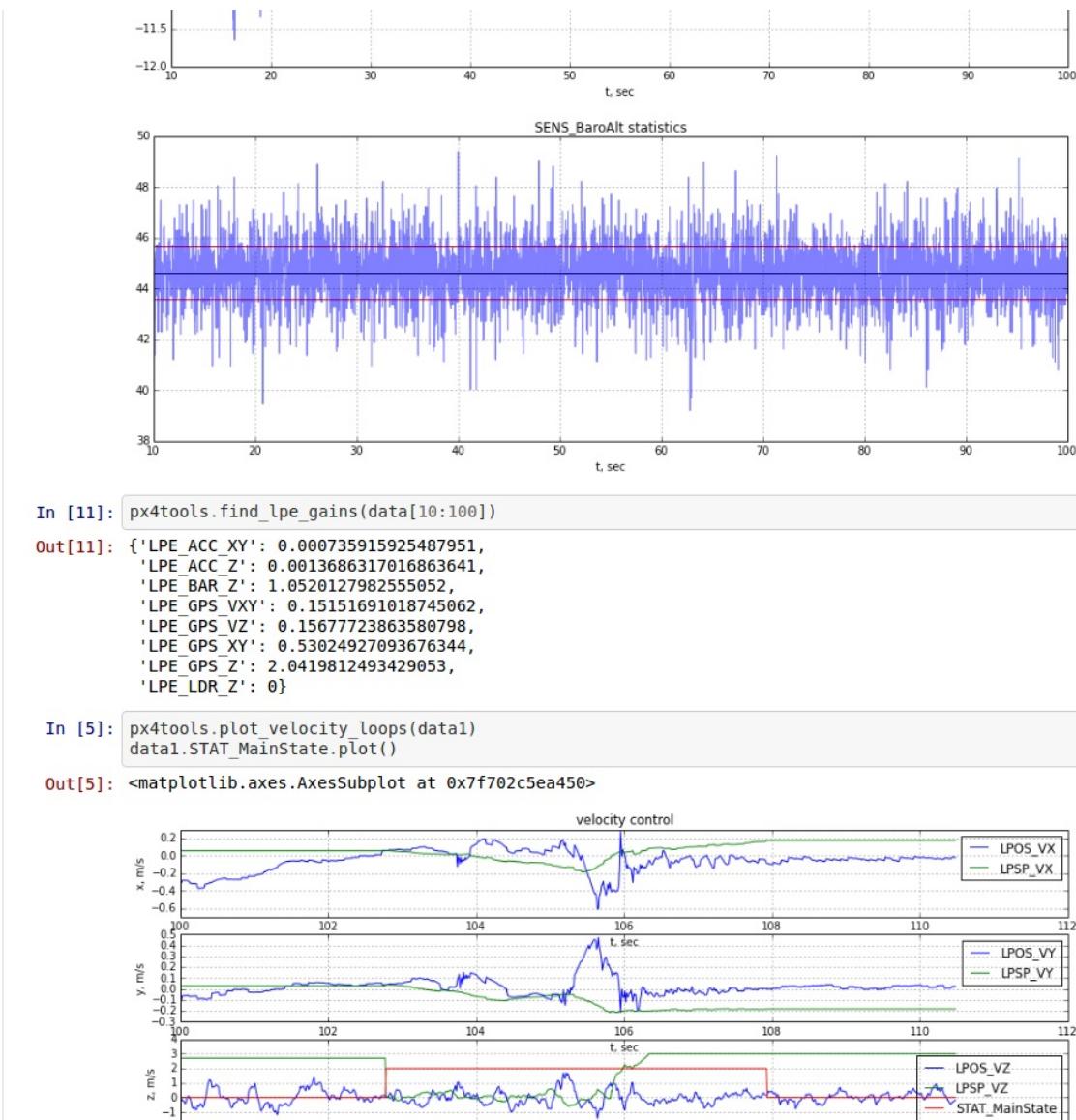
## 优点

- 基于JAVA,跨平台
- 直观的用户界面，没有编程知识的要求

## 缺点

- 分析受限于系统内置的一些特性

## PX4Tools



## 安装

- 建议的方法是使用anaconda3. 详情见 [px4tools github page](https://github.com/dronecrew/px4tools) .

```
conda install -c https://conda.anaconda.org/dronecrew px4tools
```

## 优点

- 便于分享，用户可以查看笔记在github(e.g. <https://github.com/jgoppert/lpe-analysis/blob/master/15-09-30%20Kabir%20Log.ipynb>)
- 基于python,跨平台，产品有anaconda 2 and anaconda3
- ipython/ jupyter 笔记容易分享和分析
- 高级绘图能力允许做细节的分析

## 缺点

- 要求用户懂python
- 目前要求用户在使用之前将log文件转化为csv文件

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# EKF2的Log文件回放

官网英文原文地址：[http://dev.px4.io/ekf2\\_log\\_replay.html](http://dev.px4.io/ekf2_log_replay.html)

This page shows you how you can tune the parameters of the EKF2 estimator by using the replay feature on a real flight log.

## Introduction

A developer has the possibility to enable onboard logging of the EKF2 estimator input sensor data. This allows him to tune the parameters of the estimator offline by running a replay on the logged data trying out different sets of parameters. The remainder of this page will explain which parameters have to be set in order to benefit from this feature and how to correctly deploy it.

## Prerequisites

- set the parameter **EKF2\_REC\_RPL** to 1. This will tell the estimator to publish special replay messages for logging.
- set the parameter **SDLOG\_PRIO\_BOOST** to a value contained in the set {0, 1, 2, 3}. A value of 0 means that the onboard logging app has a default (low) scheduling priority. A low scheduling priority can lead to a loss of logging messages. If you find that your log file contains 'gaps' due to skipped messages then you can increase this parameter to a maximum value of 3. Testing has shown that a minimum value of 2 is required in order to avoid loss of data.
- You may need to format the SD card just before you want to record the log file.

## Deployment

Once you have a real flight log then you can run a replay on it by using the following command in the root directory of your PX4 Firmware

```
make posix_sitl_replay replay logfile=absolute_path_to_log_file/my_log_file.px4log
```

where 'absolute\_path\_to\_log\_file/my\_log\_file.px4log' is a placeholder for the absolute path of the log file you want to run the replay on. Once the command has executed check the terminal for the location and name of the replayed log file.

## Changing tuning parameters for a replay

You can set the estimator parameter values for the replay in the file **replay\_params.txt** located in the same directory as your replayed log file, e.g.

**build\_posix\_sitl\_replay/src/firmware posix/rootfs/replay\_params.txt**. When running the replay for the first time (e.g. after a **make clean**) this file will be autogenerated and filled with the default EKF2 parameter values taken from the flight log. After that you can change any EKF2 parameter value by changing the corresponding number in the text file. Setting the noise value for the gyro bias would require the following line.

```
EKF2_GY_BIAS_NOISE 0.001
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# System-wide Replay

官网英文原文地址：<http://dev.px4.io/advanced-replay.html>

Based on ORB messages, it's possible to record and replay arbitrary parts of the system. For this to work, the new logger needs to be enabled ( `SYS_LOGGER` set to 1).

Replay is useful to test the effect of different parameter values based on real data, compare different estimators, etc.

## Prerequisites

The first thing that needs to be done is to identify the module or modules that should be replayed. Then, all the inputs to these modules, i.e. subscribed ORB topics. For system-wide replay, this consists of all hardware input: sensors, RC input, mavlink commands and file system.

All identified topics need to be logged at full rate (see [logging](#)). For `ekf2` this is already the case with the default set of logged topics.

It is important that all replayed topics contain only a single absolute timestamp, which is the automatically generated field `timestamp`. Should there be more timestamps, then they must be relative with respect to the main timestamp. For an example, see [sensor\\_combined.msg](#). Reasons for this are given below.

## Usage

- First, choose the file to replay, and build the target (from within the Firmware directory):

```
export replay=<absolute_path_to_log_file.ulg>
make posix_sitl_default
```

This will create the output in a separate build directory

`build_posix_sitl_default_replay` (so that the parameters don't interfere with normal builds). It's possible to choose any posix SITL build target for replay, the build system knows through the `replay` environment variable that it's in replay mode.

- Add ORB publisher rules file in

`build_posix_sitl_default_replay/src/firmware/posix/rootfs/orb_publisher.rules`. This file defines which module is allowed to publish which messages. It has the following

format:

```
restrict_topics: <topic1>, <topic2>, ..., <topicN>
module: <module>
ignore_others: <true/false>
```

It means that the given list of topics should only be published by `<module>` (which is the command name). Publications to any of these topics from another module are silently ignored. If `ignore_others` is `true`, then publications to other topics from `<module>` are ignored.

For replay, we only want the `replay` module to be able to publish the previously identified list of topics. So for replaying `ekf2`, the rules file looks like this:

```
restrict_topics: sensor_combined, vehicle_gps_position, vehicle_land_detected
module: replay
ignore_others: true
```

This allows that the modules, which usually publish these topics, don't need to be disabled for replay.

- Optional: setup parameter overrides in the file

`build_posix_sitl_default_replay/src/firmware/posix/rootfs/replay_params.txt`. This file should contain a list of `<param_name> <value>`, like:

```
EKF2_GB_NOISE 0.001
```

By default, all parameters from the log file are applied. When a parameter changed during recording, it will be changed as well at the right time during replay. A parameter in the `replay_params.txt` will override the value and changes to it from the log file will not be applied.

- Optional: copy `dataman` missions file from the SD card to the build directory. Only necessary if a mission should be replayed.
- Start the replay:

```
make posix_sitl_default jmavsim
```

This will automatically open the log file, apply the parameters and start to replay. Once done, it will be reported and the process can be exited. Then the newly generated log file can be analyzed, it has `_replayed` appended to its file name.

Note that the above command will show the simulator as well, but depending on what is being replayed, it will not show what's actually going on. It's possible to connect via QGC and e.g. view the changing attitude during replay.

- Finally, unset the environment variable, so that the normal build targets are used again:

```
unset replay
```

## Important Notes

- During replay, all dropouts in the log file are reported. These have a negative effect on replay and thus it should be taken care that dropouts are avoided during recording.
- It is currently only possible to replay in 'real-time', meaning as fast as the recording was done. This is planned to be extended in the future.
- A message that has a timestamp of 0 will be considered invalid and not be replayed.

## Behind the Scenes

Replay is split into 3 components:

- a replay module
- ORB publisher rules
- time handling

The replay module reads the log and publishes the messages with the same speed as they were recorded. A constant offset is added to the timestamp of each message to match the current system time (this is the reason why all other timestamps need to be relative). The command `replay tryapplyparams` is executed before all other modules are loaded and applies the parameters from the log and user-set parameters. Then as the last command, `replay trystart` will again apply the parameters and start the actual replay. Both commands do nothing if the environment variable `replay` is not set.

The ORB publisher rules allow to select which part of the system is replayed, as described above. They are only compiled for the posix SITL targets.

The **time handling** is still an **open point**, and needs to be implemented.

# 安装Intel RealSense R200的驱动

官网英文原文地址：[http://dev.px4.io/advanced-realsense\\_intel.html](http://dev.px4.io/advanced-realsense_intel.html)

This tutorial aims to give instructions on how to install the camera driver of the Intel RealSense R200 camera head in Linux environment such that the gathered images can be accessed via the Robot Operation System (ROS). The RealSense R200 camera head is depicted below:



The installation of the driver package is executed on a Ubuntu operation system (OS) that runs as a guest OS in a Virtual Box. The specifications of the host computer where the Virtual Box is running, the Virtual Box and the guest system are given below:

- Host Operation System: Windows 8
- Processor: Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz
- Virtual Box: Oracle VM. Version 5.0.14 r105127
- Extensions: Extension package for Virtual Box installed (Needed for USB3 support)
- Guest Operation System: Linux - Ubuntu 14.04.3 LTS

The tutorial is ordered in the following way: In a first part it is shown how to install Ubuntu 14.04 as a guest OS in the Virtual Box. In a second part is shown how to install ROS Indigo and the camera driver. The ensuing frequently used expressions have the following meaning:

- Virtual Box (VB): Program that runs different Virtual Machines. In this case the Oracle VM.
- Virtual Machine (VM): The operation system that runs in the Virtual Box as a guest system. In this case Ubuntu.

# Installing Ubuntu 14.04.3 LTS in Virtual Box

- Create a new Virtual Machine (VM): Linux 64-Bit.
- Download the iso file of Ubuntu 14.04.3 LTS: ([ubuntu-14.04.3-desktop-amd64.iso](#)).
- Installation of Ubuntu:
  - During the installation procedure leave the following two options unchecked:
    - Download updates while installing
    - Install this third party software
- After the installation you might need to enable the Virtual Box to display Ubuntu on the whole desktop:
  - Start VM Ubuntu and login, Click on Devices->Insert Guest Additions CD image in the menu bar of the Virtual Box.
  - Click on "Run" and enter password on the windows that pop up in Ubuntu.
  - Wait until the installation is completed and then restart. Now, it should be possible to display the VM on the whole desktop.
  - If a window pops up in Ubuntu that asks whether to update, reject to update at this point.
- Enable USB 3 Controller in Virtual Box:
  - Shut down Virtual Machine.
    - Go to the settings of the Virtual Machine to the menu selection USB and choose: "USB 3.0(xHCI)". This is only possible if you have installed the extension package for the Virtual Box.
- Start the Virtual Machine again.

# Installing ROS Indigo

- Follow instructions given at [ROS indigo installation guide](#):
  - Install Desktop-Full version.
  - Execute steps described in the sections "Initialize rosdep" and "Environment setup".

# Installing camera driver

- Install git:

```
sudo apt-get install git
```

- Download and install the driver

- Clone [RealSense\\_ROS repository](#): 【按：链接已失效】

```
git clone https://github.com/PercATI/RealSense_ROS.git
```

- Follow instructions given in [here](#). 【按：链接已失效】

- Press the enter button when the questions whether to install the following installation packages show up:

Intel Low Power Subsystem support in ACPI mode (MFD\_INTEL\_LPSS\_ACPI) [N/m/y/?] (N)

Intel Low Power Subsystem support in PCI mode (MFD\_INTEL\_LPSS\_PCI) [N/m/y/?] (N)

Dell Airplane Mode Switch driver (DELL\_RBTN) [N/m/y/?] (NEW)

- The following error message that can appear at the end of the installation process should not lead to a malfunction of the driver:

```
rmmmod: ERROR: Module uvcvideo is not currently loaded
```

- After the installation has completed, reboot the Virtual Machine.

- Test camera driver:

- Connect the Intel RealSense camera head with the computer with a USB3 cable that is plugged into a USB3 receptacle on the computer.
- Click on Devices->USB-> Intel Corp Intel RealSense 3D Camera R200 in the menu bar of the Virtual Box, in order to forward the camera USB connection to the Virtual Machine.
- Execute the file [unpacked folder]/Bin/DSReadCameraInfo:
  - If the following error message appears, unplug the camera (physically unplug USB cable from the computer). Plug it in again + Click on Devices->USB-> Intel Corp Intel RealSense 3D Camera R200 in the menu bar of the Virtual Box again and execute again the file [unpacked folder]/Bin/DSReadCameraInfo.

```
DSAPI call failed at ReadCameraInfo.cpp:134!
```

- If the camera driver works and recognises the Intel RealSense R200, you should

see specific information about the Intel RealSense R200 camera head.

- Installation and testing of the ROS nodlet:

- Follow the installation instructions in the "Installation" section given [here](#), to install the ROS nodlet.
- Follow the instructions in the "Running the R200 nodelet" section given [here](#), to test the ROS nodlet together with the Intel RealSense R200 camera head.
  - If everything works, the different data streams from the Intel RealSense R200 camera are published as ROS topics.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22

12:56:03

# 设置云台控制

官网英文原文地址：<http://dev.px4.io/advanced-gimbal-control.html>

PX4包含一个通用的安装/云台控制驱动程序，具有不同的输入和输出模式。可以选择任何输入模式来驱动任何输出。

首先，确认驱动运行，运行 `vmount start`，然后配置其参数。

## 参数

参数描述在 `src/drivers/vmount/vmount_params.c` 中。其中，最重要的参数是输入 (`MNT_MODE_IN`) 和输出 (`MNT_MODE_OUT`) 模式。默认情况下，禁用输入。可以选择任何输入方式来驱动任何可用的输出。如果选择了 mavlink 输入模式，则可以另外启动手动 RC 输入 (`MNT_MAN_CONTROL`)。只要没有收到 mavlink 消息，或 mavlink 明确请求 RC 模式，参数都是有用的。

## 配置云台混控器的AUX输出

云台使用控制组#2(请参阅混控和执行器篇)，这是混控器设置：

```
roll
M: 1
O: 10000 10000 0 -10000 10000
S: 2 0 10000 10000 0 -10000 10000

pitch
M: 1
O: 10000 10000 0 -10000 10000
S: 2 1 10000 10000 0 -10000 10000

yaw
M: 1
O: 10000 10000 0 -10000 10000
S: 2 2 10000 10000 0 -10000 10000
```

将所需要的配置添加到主混控器或者辅混控器。

## 测试

驱动程序提供了一些简单的测试命令。需要先运行 `vmount stop` 停止。以下描述了SITL中的测试，但是这些命令行也可在真是设备上运行。

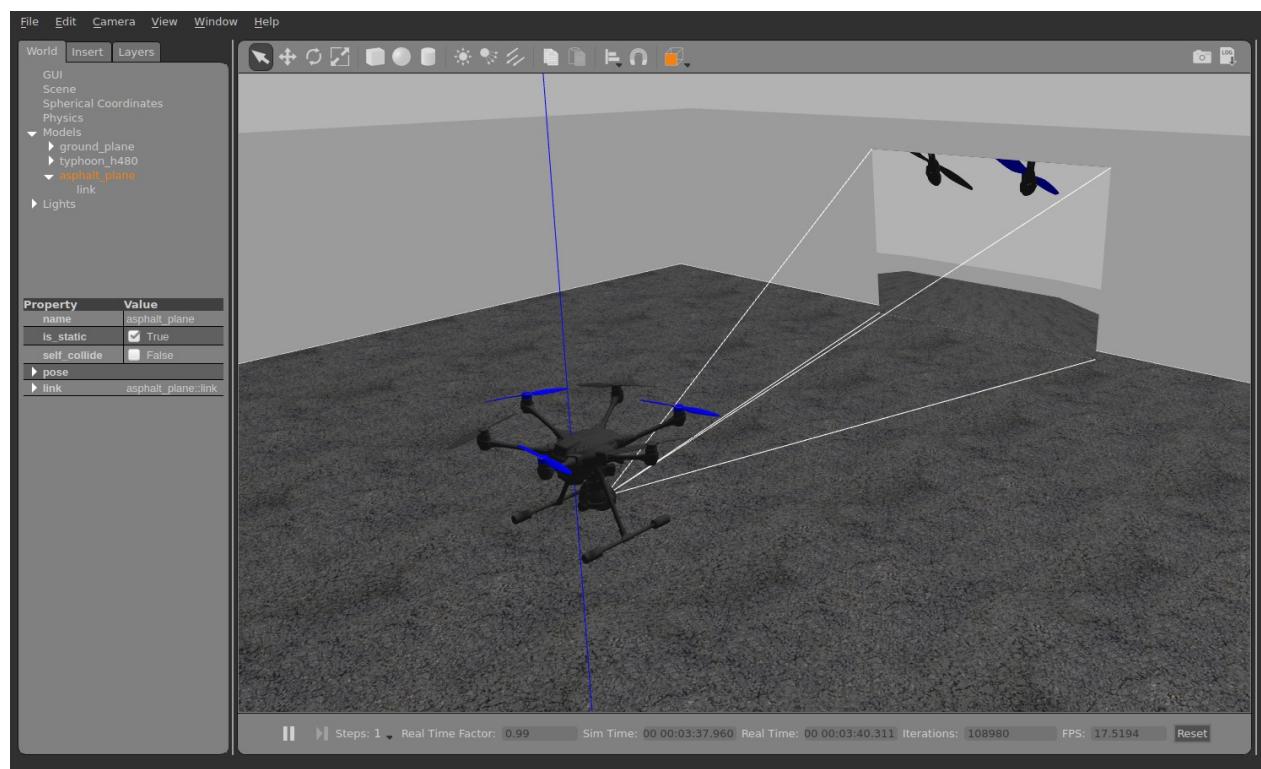
开启仿真(无需为此更改参数):

```
make posix gazebo_typhoon_h480
```

先确认已开桨，比如使用命令 `commander takeoff`，然后运行：

```
vmount test yaw 30
```

以控制云台。请注意，仿真中云台本身会增稳，因此，如果你发送mavlink命令，请将 `stabilize` 标志位置为`false`。



© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22

12:56:03

# 切换状态估计器

官网英文原文地址：[http://dev.px4.io/advanced-switching\\_state\\_estimators.html](http://dev.px4.io/advanced-switching_state_estimators.html)

本文主要介绍PX4中有哪些可用的状态估计器以及用户该如何在不同的估计器间进行切换。

## 可用的估计器

### 1. Q attitude estimator(四元数姿态估计)

四元数姿态估计方法非常简单，就是基于四元数的姿态互补滤波器。

### 2. INAV position estimator(惯导位置估计)

惯导位置估计使用互补滤波器对三维位置以及速度进行估计。。

### 3. LPE position estimator(LPE位置估计)

LPE (Local Position Estimator) 位置估计使用扩展卡尔曼滤波器对三维位置以及速度进行估计。

### 4. EKF2 attitude, position and wind states estimator (EKF2姿态，位置以及风速估计)

EKF2使用扩展卡尔曼滤波器进行三维的姿态，位置/速度以及风的状态进行估计。

### 5. EKF attitude, position and wind states estimator (depricated)(EKF姿态，位置以及风速估计(已过时)) (即固件参数列表中的Attitude EKF estimator和Position Estimator)

这是一个类似于EKF2的扩展卡尔曼滤波器。然而，很快它就将完全由EKF2代替。

此滤波器仅用于固定翼。

## 如何使能不同的估计器

对于多旋翼和垂直起降飞行器，使用参数**SYS\_MC\_EST\_GROUP**在下列配置中进行选择。

目前只有已过时的EKF估计器被用于非垂直起降的飞行器。它将很快被EKF2代替。

<b>SYS_MC_EST_GROUP</b>	<b>Q Estimator</b>	<b>INAV</b>	<b>LPE</b>	<b>EKF2</b>
0	使能	使能		
1	使能		使能	
2				使能

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# 外部模块

官网英文原文地址：<http://dev.px4.io/advanced-out-of-tree-modules.html>

本教程描述了向PX4构建中添加外部模块的可能性。

外部模块可以使用与内部模块相同的模块，并且可以通过\$mu\$ORB与内部模块交互

## 使用

- `EXTERNAL_MODULES_LOCATION` 需要指向一个与原生固件Firmware具有相同结构的目录(因此需包含一个称为 `src` 的目录)。
- 有两种方法：将现有模块(如`examples/px4_simple_app`)复制到外部目录，或者直接创建一个新的模块。
- 重命名模块(包括CMakeLists.txt中的 `MODULE` )或将其从现有的Firmware/cmake//config中移除。这是为了避免与内部模块发生冲突。
- 添加一个文件 `$EXTERNAL_MODULES_LOCATION/CMakeLists.txt`，其内容包括：

```
set(config_module_list_external
 modules/<new_module>
 PARENT_SCOPE
)
```

- 添加一行 `EXTERNAL` 到 `modules/<new_module>/CMakeLists.txt` 下的 `px4_add_module` 函数中，例如像这样：

```
px4_add_module(
 MODULE modules__test_app
 MAIN test_app
 STACK_MAIN 2000
 SRCS
 px4_simple_app.c
 DEPENDS
 platforms__common
 EXTERNAL
)
```

- 执行 `make posix EXTERNAL_MODULES_LOCATION=<path>`。可以使用任何其他的构建目标，但是构建目录必须是不存在的。如果它已经存在，你也可以在build文件夹中设置cmake变量。对于以后要增加的构建，就不需要再指定 `EXTERNAL_MODULES_LOCATION` 了。

12:56:03

# ULog File Format

官网英文原文地址：<http://dev.px4.io/advanced-ulog-file-format.html>

ULog is the file format used for logging system data. The format is self-describing, i.e. it contains the format and message types that are logged.

It can be used for logging device inputs (sensors, etc.), internal states (cpu load, attitude, etc.) and printf log messages.

The format uses Little Endian for all binary types.

## Data types

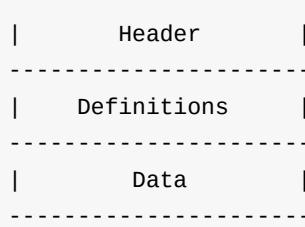
The following binary types are used. They all correspond to the types in C:

Type	Size in Bytes
int8_t, uint8_t	1
int16_t, uint16_t	2
int32_t, uint32_t	4
int64_t, uint64_t	8
float	4
double	8
bool, char	1

Additionally all can be used as an array, eg. `float[5]`. In general all strings (`char[length]`) do not contain a `'\0'` at the end. String comparisons are case sensitive.

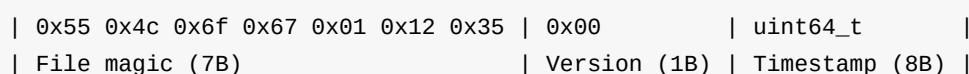
## File structure

The file consists of three sections:



## Header Section

The header is a fixed-size section and has the following format (16 bytes):



Version is the file format version, currently 0. Timestamp is an `uint64_t` integer, denotes the start of the logging in microseconds.

## Definitions Section

Variable length section, contains version information, format definitions, and (initial) parameter values.

The Definitions and Data sections consist of a stream of messages. Each starts with this header:

```

struct message_header_s {
 uint16_t msg_size;
 uint8_t msg_type
};

```

`msg_size` is the size of the message in bytes without the header (`hdr_size = 3 bytes`). `msg_type` defines the content and is one of the following:

- 'F': format definition for a single (composite) type that can be logged or used in another definition as a nested type.

```
struct message_format_s {
 struct message_header_s header;
 char format[header.msg_size-hdr_size];
};
```

`format` : plain-text string with the following format: `message_name:field0;field1;`

There can be an arbitrary amount of fields (at least 1), separated by `;`. A field has the format: `type field_name` or `type[array_length] field_name` for arrays (only fixed size arrays are supported). `type` is one of the basic binary types or a `message_name` of another format definition (nested usage). A type can be used before it's defined. There can be arbitrary nesting but no circular dependencies.

Some field names are special:

- `timestamp` : every logged message (`message_add_logged_s`) must include a timestamp field (does not need to be the first field). Its type can be: `uint64_t` (currently the only one used), `uint32_t`, `uint16_t` or `uint8_t`. The unit is always microseconds, except for `uint8_t` it's milliseconds. A log writer must make sure to log messages often enough to be able to detect wrap-arounds and a log reader must handle wrap-arounds (and take into account dropouts). The timestamp must always be monotonic increasing for a message serie with the same `msg_id`.
- Padding: field names that start with `_padding` should not be displayed and their data must be ignored by a reader. These fields can be inserted by a writer to ensure correct alignment.

If the padding field is the last field, then this field will not be logged, to avoid writing unnecessary data. This means the `message_data_s.data` will be shorter by the size of the padding. However the padding is still needed when the message is used in a nested definition.

- 'I': information message.

```
struct message_info_s {
 struct message_header_s header;
 uint8_t key_len;
 char key[key_len];
 char value[header.msg_size-hdr_size-1-key_len]
};
```

`key` is a plain string, as in the format message, but consists of only a single field without ending `;`, eg. `float[3] myvalues . value` contains the data as described by `key`.

Predefined information messages are:

key	Description	Example for value
<code>char[value_len] sys_name</code>	Name of the system	"PX4"
<code>char[value_len] ver_hw</code>	Hardware version	"PX4FMU_V4"
<code>char[value_len] ver_sw</code>	Software version (git tag)	"7f65e01"
<code>uint32_t ver_sw_release</code>	Software version (see below)	0x010401ff
<code>char[value_len] sys_os_name</code>	Operating System Name	"Linux"
<code>char[value_len] sys_os_ver</code>	OS version (git tag)	"9f82919"
<code>uint32_t ver_os_release</code>	OS version (see below)	0x010401ff
<code>char[value_len] sys_toolchain</code>	Toolchain Name	"GNU GCC"
<code>char[value_len] sys_toolchain_ver</code>	Toolchain Version	"6.2.1"
<code>char[value_len] sys_mcu</code>	Chip name and revision	"STM32F42x, rev A"
<code>char[value_len] sys_uuid</code>	Unique identifier for vehicle (eg. MCU ID)	"392a93e32fa3"...
<code>char[value_len] replay</code>	File name of replayed log if in replay mode	"log001.ulg"
<code>int32_t time_ref_utc</code>	UTC Time offset in seconds	-3600

The format of `ver_sw_release` and `ver_os_release` is: 0xAABBCCTT, where AA is major, BB is minor, CC is patch and TT is the type. Type is defined as following: `>= 0` : development, `>= 64` : alpha version, `>= 128` : beta version, `>= 192` : RC version, `= 255` : release version. So for example 0x010402ff translates into the release version v1.4.2.

- 'P': parameter message. Same format as `message_info_s`. If a parameter dynamically changes during runtime, this message can also be used in the Data section. The data type is restricted to: `int32_t`, `float`.

This section ends before the start of the first `message_add_logged_s` message.

## Data Section

The following messages belong to this section:

- 'A': subscribe a message by name and give it an id that is used in `message_data_s`. This must come before the first corresponding `message_data_s`.

```
struct message_add_logged_s {
 struct message_header_s header;
 uint8_t multi_id;
 uint16_t msg_id;
 char message_name[header.msg_size-hdr_size-3];
};
```

`multi_id` : the same message format can have multiple instances, given by `multi_id`. The default and first instance is 0.

`msg_id` : unique id to match `message_data_s` data. The first use must set this to 0, then increase it. The same `msg_id` must not be used twice for different subscriptions, not even after unsubscribing.

`message_name` : message name to subscribe to. Must match one of the `message_format_s` definitions.

- 'R': unsubscribe a message, to mark that it will not be logged anymore (not used currently).

```
struct message_remove_logged_s {
 struct message_header_s header;
 uint16_t msg_id;
};
```

- 'D': contains logged data.

```
struct message_data_s {
 struct message_header_s header;
 uint16_t msg_id;
 uint8_t data[header.msg_size-hdr_size];
};
```

`msg_id` : as defined by a `message_add_logged_s` message. `data` contains the logged binary message as defined by `message_format_s`. See above for special treatment of padding fields.

- 'L': Logged string message, i.e. printf output.

```
struct message_logging_s {
 struct message_header_s header;
 uint8_t log_level;
 uint64_t timestamp;
 char message[header.msg_size-hdr_size-9]
};
```

`timestamp` : in microseconds, `log_level` : same as in the Linux kernel:

Name	Level value	Meaning
EMERG	'0'	System is unusable
ALERT	'1'	Action must be taken immediately
CRIT	'2'	Critical conditions
ERR	'3'	Error conditions
WARNING	'4'	Warning conditions
NOTICE	'5'	Normal but significant condition
INFO	'6'	Informational
DEBUG	'7'	Debug-level messages

- 'S': synchronization message so that a reader can recover from a corrupt message by search for the next sync message (not used currently).

```
struct message_sync_s {
 struct message_header_s header;
 uint8_t sync_magic[8];
};
```

`sync_magic` : to be defined.

- 'O': mark a dropout (lost logging messages) of a given duration in ms. Dropouts can occur eg if the device is not fast enough.

```
struct message_dropout_s {
 struct message_header_s header;
 uint16_t duration;
};
```



# Licenses

官网英文原文地址：<http://dev.px4.io/advanced-licenses.html>

This page documents the licenses of various components in the system.

- [PX4 Flight Stack](#) — BSD
- [PX4 Middleware](#) — BSD
- [Pixhawk Hardware](#) — CC-BY-SA 3.0
- [Snapdragon Hardware](#) — Qualcomm Proprietary

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22  
12:56:03

# 更新软件

官网英文原文地址：[http://dev.px4.io/software\\_update.html](http://dev.px4.io/software_update.html)

The method to update the PX4 software on the drone depends on the hardware platform. For microcontroller based applications new Firmware is flashed through USB or serial. 更新无人机上面的PX4软件的方法依据硬件不同而不同。对于基于微控制器，通过USB或者是串口来烧写固件。

## 下层构造（**Infrastructure**）

### 烧写 **Bootloader**

```
(gdb) tar ext /dev/serial/by-id/usb-Black_Sphere_XXX-if00
(gdb) mon swdp_scan
(gdb) attach 1
(gdb) load tapv1_b1.elf
...
Transfer rate: 17 KB/sec, 828 bytes/write.
(gdb) kill
```

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22

12:56:03

# STM32 BootLoader

官网英文原文地址：[http://dev.px4.io/stm32\\_bootloader.html](http://dev.px4.io/stm32_bootloader.html)

PX4 bootloader的代码在Github的 [Bootloader](#) 仓库。

## 支持的飞控板

- FMUv1 (PX4FMU, STM32F4)
- FMUv2 (Pixhawk 1, STM32F4)
- FMUv3 (Pixhawk 2, STM32F4)
- FMUv4 (Pixracer 3 和 Pixhawk 3 Pro, STM32F4)
- FMUv5 (Pixhawk 4, STM32F7)
- TAPv1 (TBA, STM32F4)
- ASCv1 (TBA, STM32F4)

## 构建Bootloader

```
git clone https://github.com/PX4/Bootloader.git
cd Bootloader
make
```

经过这一步会为所有支持的飞控板生成一系列elf文件，这些文件都在BootLoader目录中。

## 刷Bootloader

重要提醒：对于一些飞控板来说，为了使用JTAG\SWD接口需要采取正确的供电顺序。正是按照所描述的这些步骤。下列的说明适用于Blackmagic\Dronecode探针。其他的JTAG探针可能需要使用类似的不同顺序。尝试刷BootLoader的开发者应该具备相关知识。如果你不知道如何进行这些操作，或许你应该再三考虑你是否确实需要更改BootLoader中的任何东西。

- 断开JTAG连线
- 连接USB电源线
- 连接JTAG

# 使用正确的串口

- LINUX: /dev/serial/by-id/usb-Black\_Sphere\_XXX-if00
- MAC OS: 确认使用的是xxx口而不是tty.xxx口: tar ext /dev/tty.usbmodemDDEasdf

```
arm-none-eabi-gdb
(gdb) tar ext /dev/serial/by-id/usb-Black_Sphere_XXX-if00
(gdb) mon swdp_scan
(gdb) attach 1
(gdb) mon option erase
(gdb) mon erase_mass
(gdb) load tapv1_bl.elf
...
Transfer rate: 17 KB/sec, 828 bytes/write.
(gdb) kill
```

## J-Link

关于 [J-Link GDB server](#)的教程点进去就行了。

## 必备条件

从[Segger官网](#)下载[J-Link](#) 软件并按照其教程进行安装。

## 运行JLink GDB

FMUv1:

```
JLinkGDBServer -select USB=0 -device STM32F405RG -if SWD-DP -speed 20000
```

AeroFC:

```
JLinkGDBServer -select USB=0 -device STM32F429AI -if SWD-DP -speed 20000
```

## 连接GDB

```
arm-none-eabi-gdb
(gdb) tar ext :2331
(gdb) load aerofcv1_bl.elf
```

## 故障检测

如果上述任意一条指令没有找到，要么你就是没有使用Blackmagic探针，或者是软件过时了。首先尝试升级探针软件。

如果出现这个错误：`Error erasing flash with vFlashErase packet`

断开目标连接（同时让JTAG保持连接），进而运行下列指令

```
mon tpwr disable
swdp_scan
attach 1
load tapv1.bl.elf
```

此举将禁用目标连接的电源并尝试另一个刷写循环。

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22  
12:56:03

# Testing and Continuous Integration

官网英文原文地址：<http://dev.px4.io/testing-and-ci.html>

PX4提供大量的测试和持续集成。本页提供概述。

## 在本地机器上测试

下面这条命令足够打开一个带有运行中的PX4端口的新shell。

```
make posix_sitl_shell none
```

shell可以用这个例子运行单元测试：

```
pxh> tests mixer
```

另一种选择也可以从bash中运行以下命令运行完整的单元测试：

```
make tests
```

## 在云端测试和持续集成

© PX4WIKI team all right reserved，powered by Gitbook该文件修订时间：2017-01-22  
12:56:03

# PX4 Docker Containers

官网英文原文地址：<http://dev.px4.io/advanced-docker.html>

Docker containers are available that contain the complete PX4 development toolchain including Gazebo and ROS simulation:

- **px4io/px4-dev**: toolchain including simulation
- **px4io/px4-dev-ros**: toolchain including simulation and ROS (incl. MAVROS)

Pull one of the tagged images if you're after a container that just works, for instance `px4io/px4-dev-ros:v1.0`, the `latest` container is usually changing a lot.

Dockerfiles and README can be found here:

<https://github.com/PX4/containers/tree/master/docker/px4-dev>

They are build automatically on Docker Hub: <https://hub.docker.com/u/px4io/>

## Prerequisites

Install Docker from here <https://docs.docker.com/installation/>, preferably use one of the Docker-maintained package repositories to get the latest version.

Containers are currently only supported on Linux. If you don't have Linux you can run the container inside a virtual machine, see further down for more information. Do not use `boot2docker` with the default Linux image because it contains no X-Server.

## Use the Docker container

The following will run the Docker container including support for X forwarding which makes the simulation GUI available from inside the container. It also maps the directory `<local_src>` from your computer to `<container_src>` inside the container. Please see the Docker docs for more information on volume and network port mapping.

With the `--privileged` option it will automatically have access to the devices on your host (e.g. a joystick and GPU). If you connect/disconnect a device you have to restart the container.

```
enable access to xhost from the container
xhost +

docker run -it --privileged \
-v <local_src>:<container_src>:rw \
-v /tmp/.X11-unix:/tmp/.X11-unix:ro \
-e DISPLAY=:0 \
--name=container_name px4io/px4-dev bash
```

If everything went well you should be in a new bash shell now. Verify if everything works by running SITL for example:

```
cd <container_src>
make posix_sitl_default gazebo
```

If you exit the container, your changes are left in this container. The above “docker run” command can only be used to create a new container. To get back into this container simply do:

```
start the container
sudo docker start container_name
open a new bash shell in this container
sudo docker exec -it container_name bash
```

If you need multiple shells connected to the container, just open a new shell and execute that last command again.

## Virtual machine support

Any recent Linux distribution should work.

The following configuration is tested:

- OS X with VMWare Fusion and Ubuntu 14.04 (Docker container with GUI support on Parallels make the X-Server crash).

### Memory

Use at least 4GB memory for the virtual machine.

### Compilation problems

If compilation fails with errors like this:

```
The bug is not reproducible, so it is likely a hardware or OS problem.
c++: internal compiler error: Killed (program cc1plus)
```

Try disabling parallel builds.

## Allow Docker Control from the VM Host

Edit `/etc/default/docker` and add this line:

```
DOCKER_OPTS="${DOCKER_OPTS} -H unix:///var/run/docker.sock -H 0.0.0.0:2375"
```

You can then control docker from your host OS:

```
export DOCKER_HOST=tcp://<ip of your VM>:2375
run some docker command to see if it works, e.g. ps
docker ps
```

## Legacy

The ROS multiplatform containers are not maintained anymore:

<https://github.com/PX4/containers/tree/master/docker/ros-indigo>

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03

# PX4 Continuous Integration

官网英文原文地址：<http://dev.px4.io/advanced-ci.html>

PX4 builds and testing are spread out over multiple continuous integration services.

## Travis-ci

Travis-ci is responsible for the official stable/beta/development binaries that are flashable through [QGroundControl](#). It currently uses GCC 4.9.3 included in the docker image [px4io/px4-dev-base](#) and compiles px4fmu-{v1, v2, v4}, mindpx-v2, tap-v1 with makefile target qgc\_firmware.

Travis-ci also has a MacOS posix sitl build which includes testing.

## Semaphore

Semaphore is primarily used to compile changes for the Qualcomm Snapdragon platform, but also serves as a backup to Travis-ci using the the same [px4io/px4-dev-base](#) docker image. In addition to the set of firmware compiled by Travis-ci, Semaphore also builds for the stm32discovery, crazyflie, runs unit testing, and verifies code style.

## CircleCI

CircleCI tests the proposed next version of GCC to be used for stable firmware releases using the docker image [px4io/px4-dev-nuttx-gcc\\_next](#). It uses the makefile target quick\_check which compiles px4fmu-v4\_default, posix\_sitl\_default, runs testing, and verifies code style.

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间：2017-01-22  
12:56:03

# Jenkins CI

官网英文原文地址：<http://dev.px4.io/advanced-jenkins-ci.html>

Jenkins continuous integration server on [SITL01](#) is used to automatically run integration tests against PX4 SITL.

## Overview

- Involved components: Jenkins, Docker, PX4 POSIX SITL
- Tests run inside [Docker Containers](#)
  - Jenkins executes 2 jobs: one to check each PR against master, and the other to check every push on master

## Test Execution

Jenkins uses [run\\_container.bash](#) to start the container which in turn executes [run\\_tests.bash](#) to compile and run the tests.

If Docker is installed the same method can be used locally:

```
cd <directory_where_firmware_is_cloned>
sudo WORKSPACE=$(pwd) ./Firmware/integrationtests/run_container.bash
```

## Server Setup

### Installation

See setup [script/log](#) for details on how Jenkins got installed and maintained.

### Configuration

- Jenkins security enabled
- Installed plugins
  - [github](#)
  - [github pull request builder](#)
  - [embeddable build status plugin](#)

- s3 plugin
- notification plugin
- collapsing console sections
- postbuildscript

© PX4WIKI team all right reserved , powered by Gitbook 该文件修订时间 : 2017-01-22  
12:56:03